CMSC 475/675 Neural Networks

Lecture 2: Neural Networks

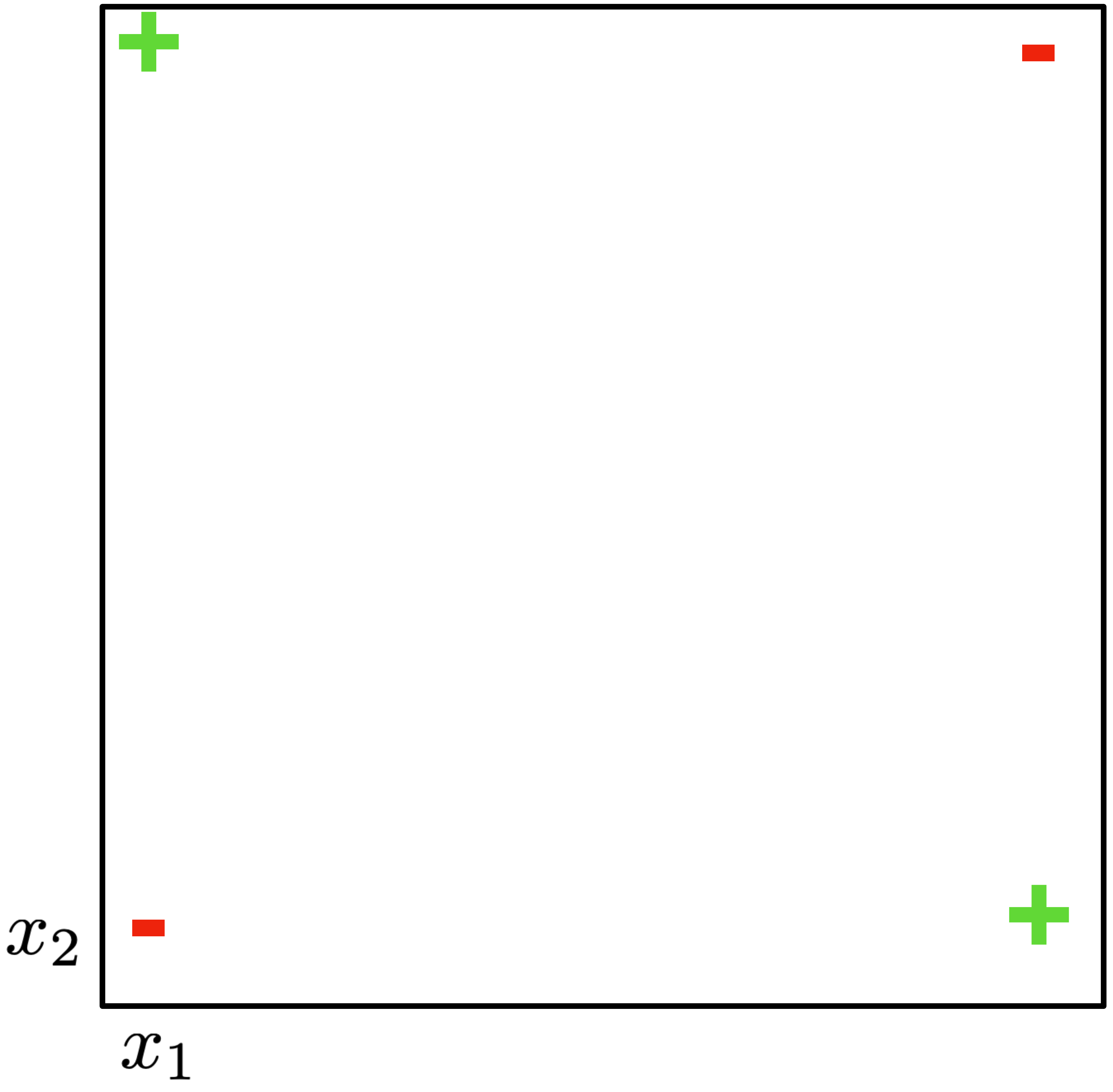
Some slides from Suren Jayasuriya (ASU), Phillip Isola (MIT)

UMBC

Artificial Intelligence

$$\hat{y} = \boldsymbol{w}^\top \boldsymbol{x} + b$$

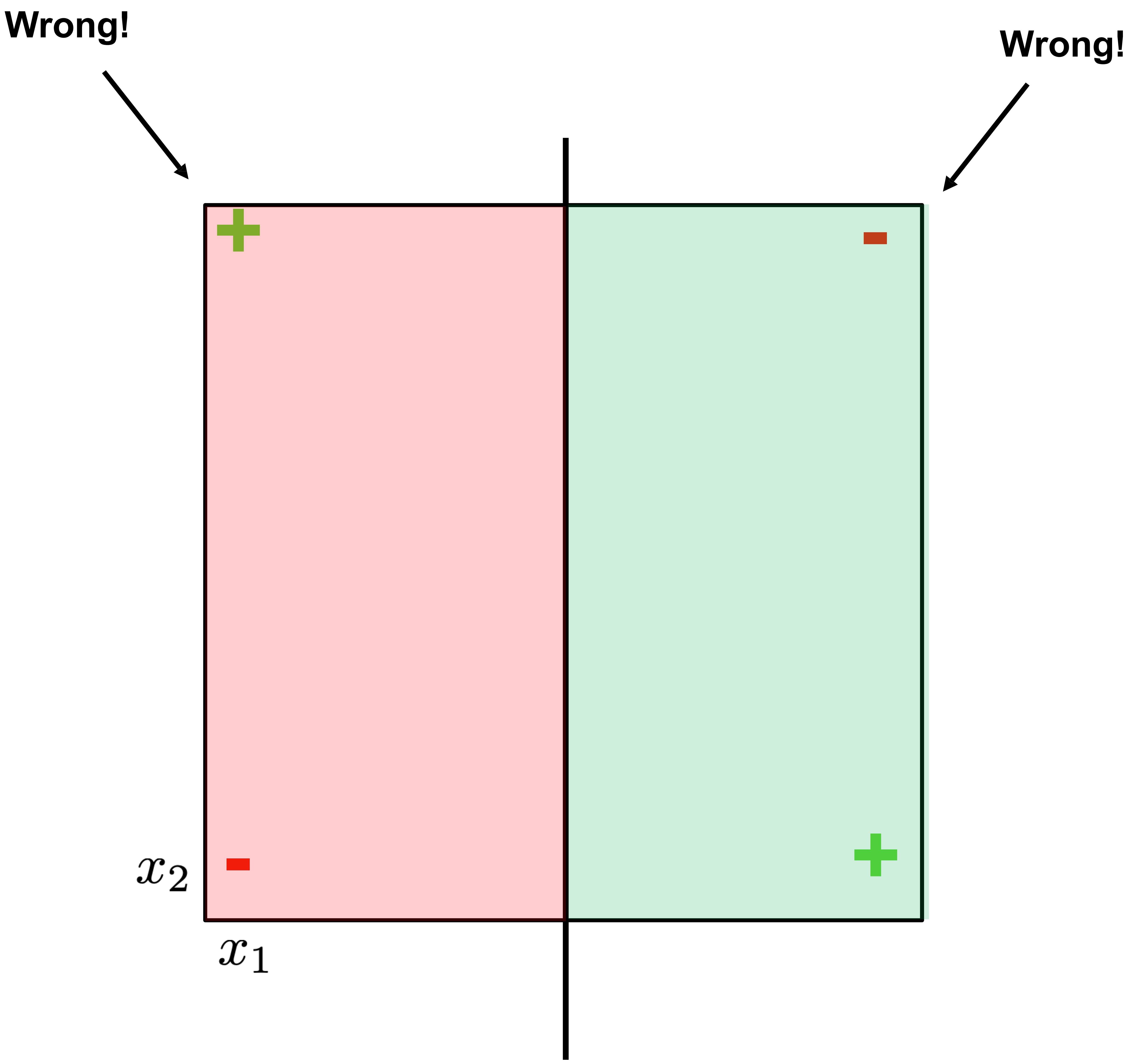
Limitations to linear classifiers



		x_2	
		0	1
x_1	0	0	1
	1	1	0

XOR

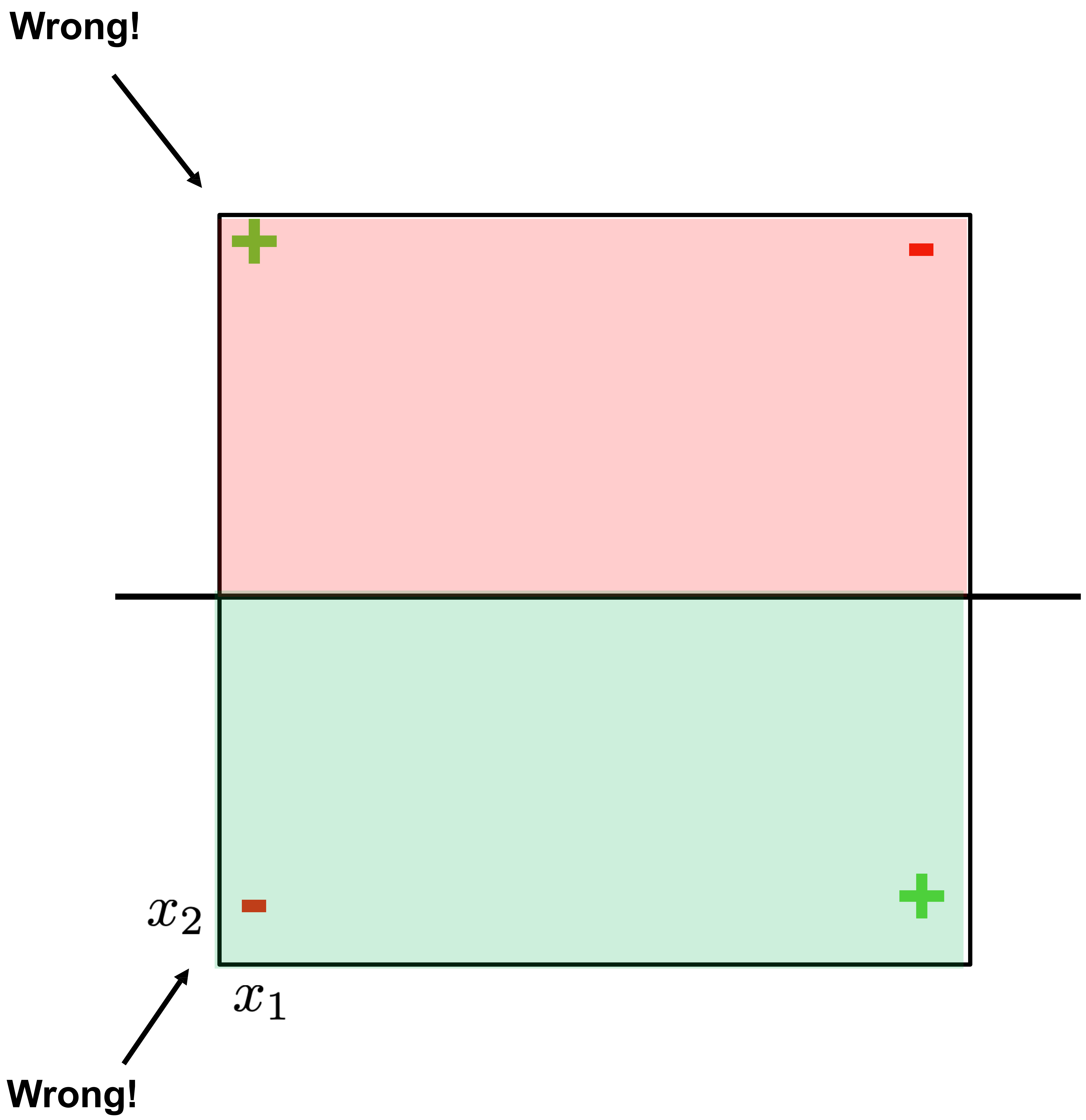
Limitations to linear classifiers



		x_2	
		0	1
x_1	0	0	1
	1	1	0

XOR

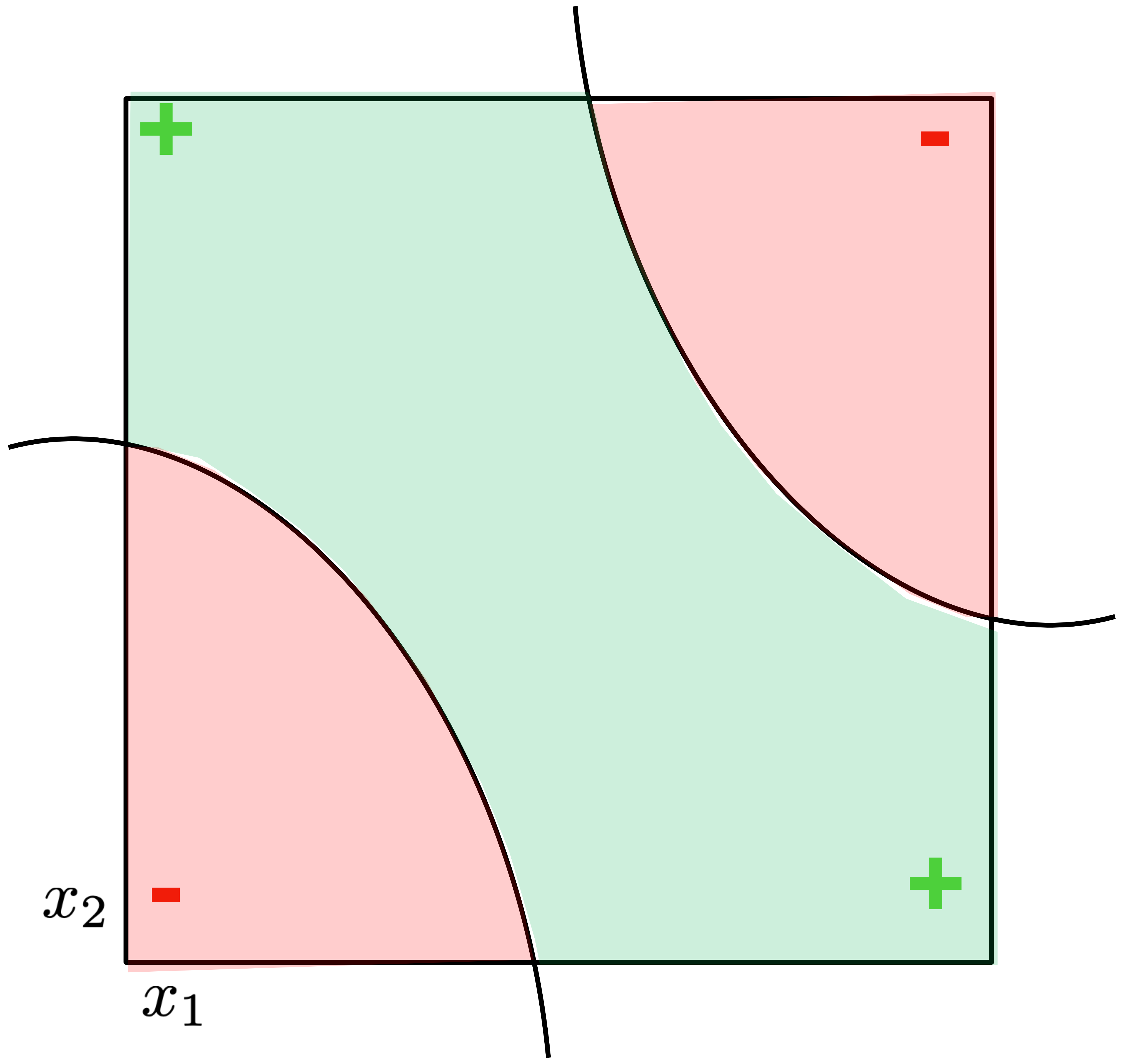
Limitations to linear classifiers



		x_2	
		0	1
x_1	0	0	1
	1	1	0

XOR

Goal: Non-linear decision boundary

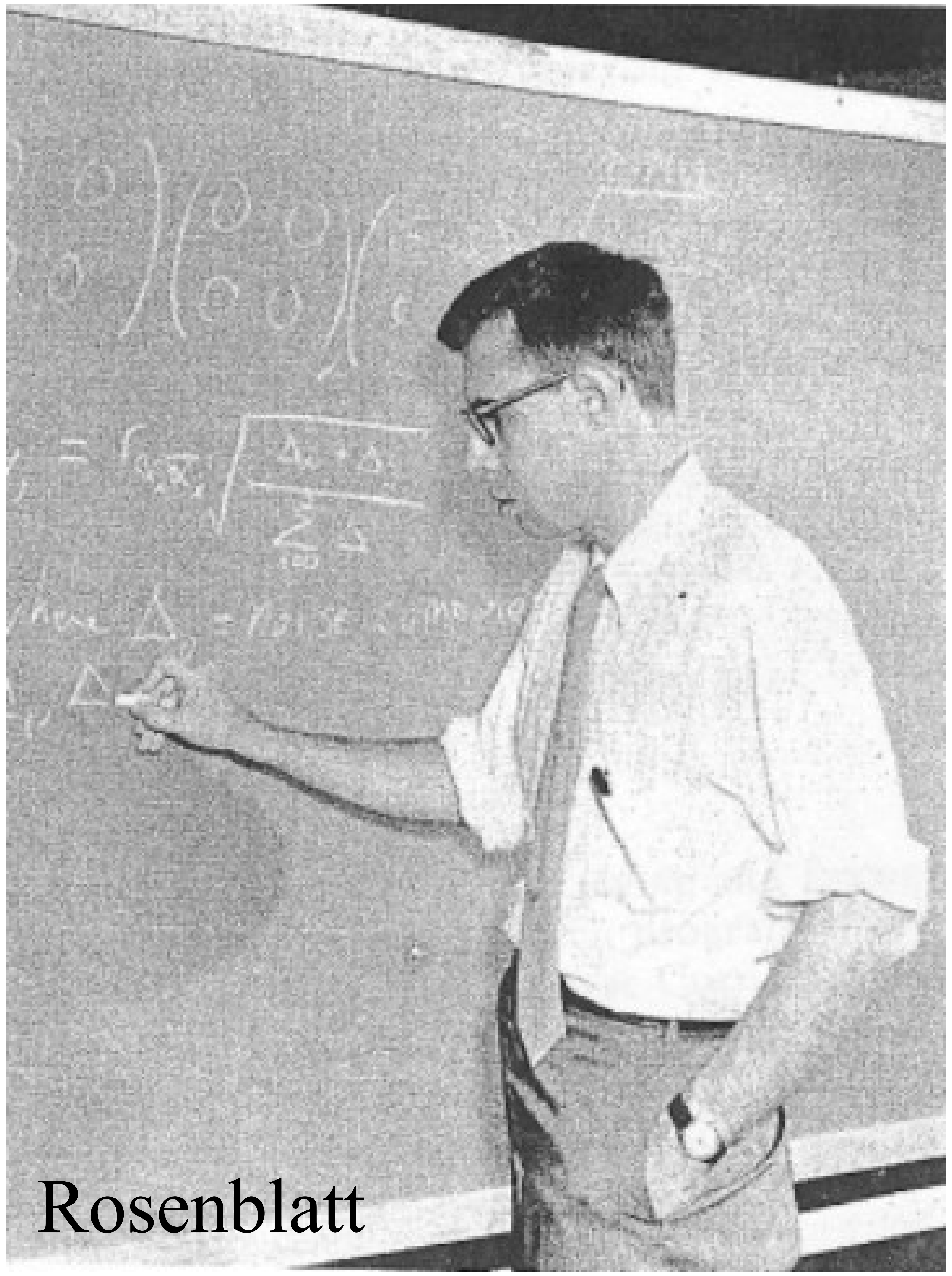


		x_2	
		0	1
x_1	0	0	1
	1	1	0

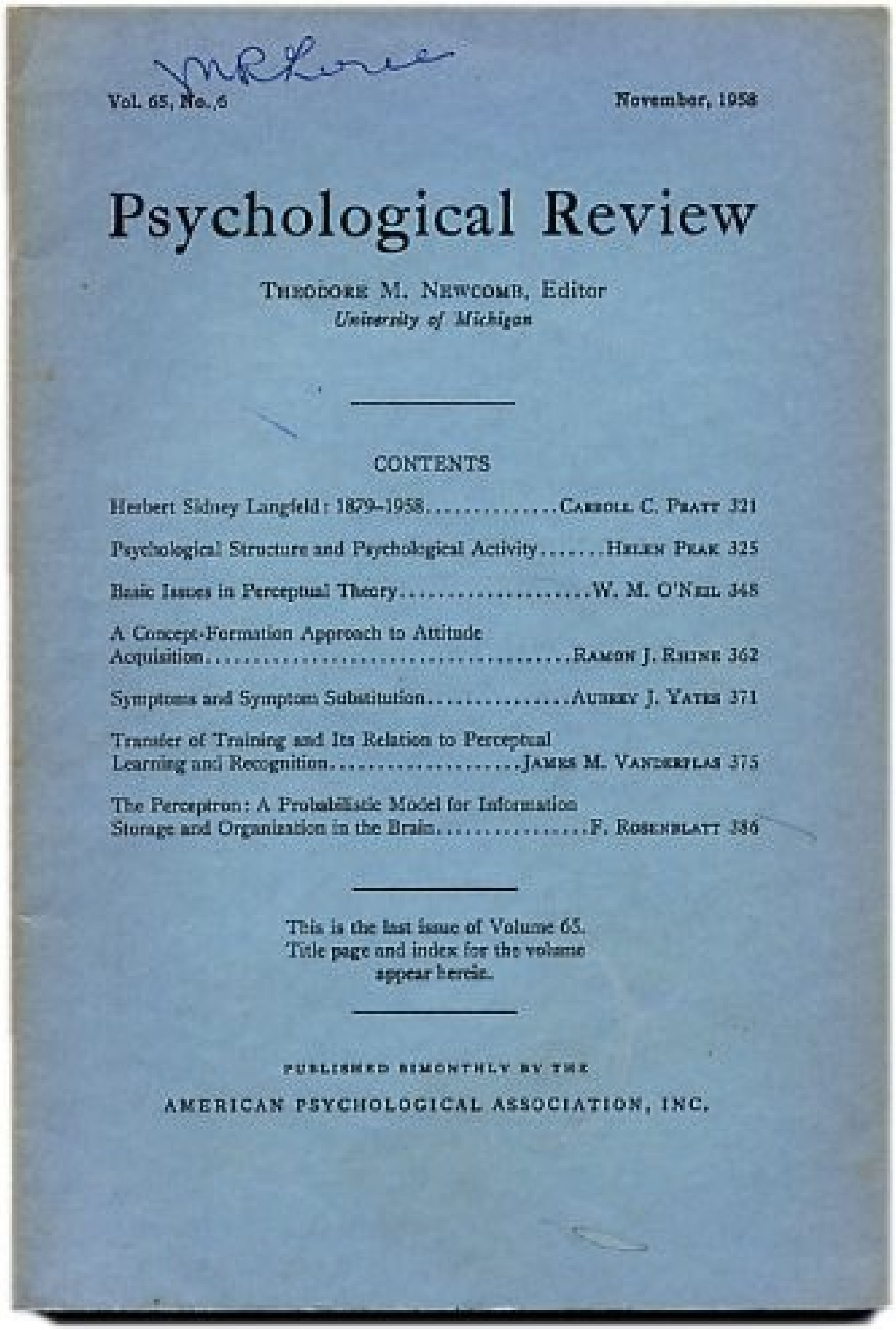
XOR

A brief history of Neural Networks

Perceptrons, 1958



http://www.ecse.rpi.edu/homepages/nagy/PDF_chrono/2011_Nagy_Pace_FR.pdf. Photo by George Nagy



MRLoree

Vol. 65, No. 6 November, 1958

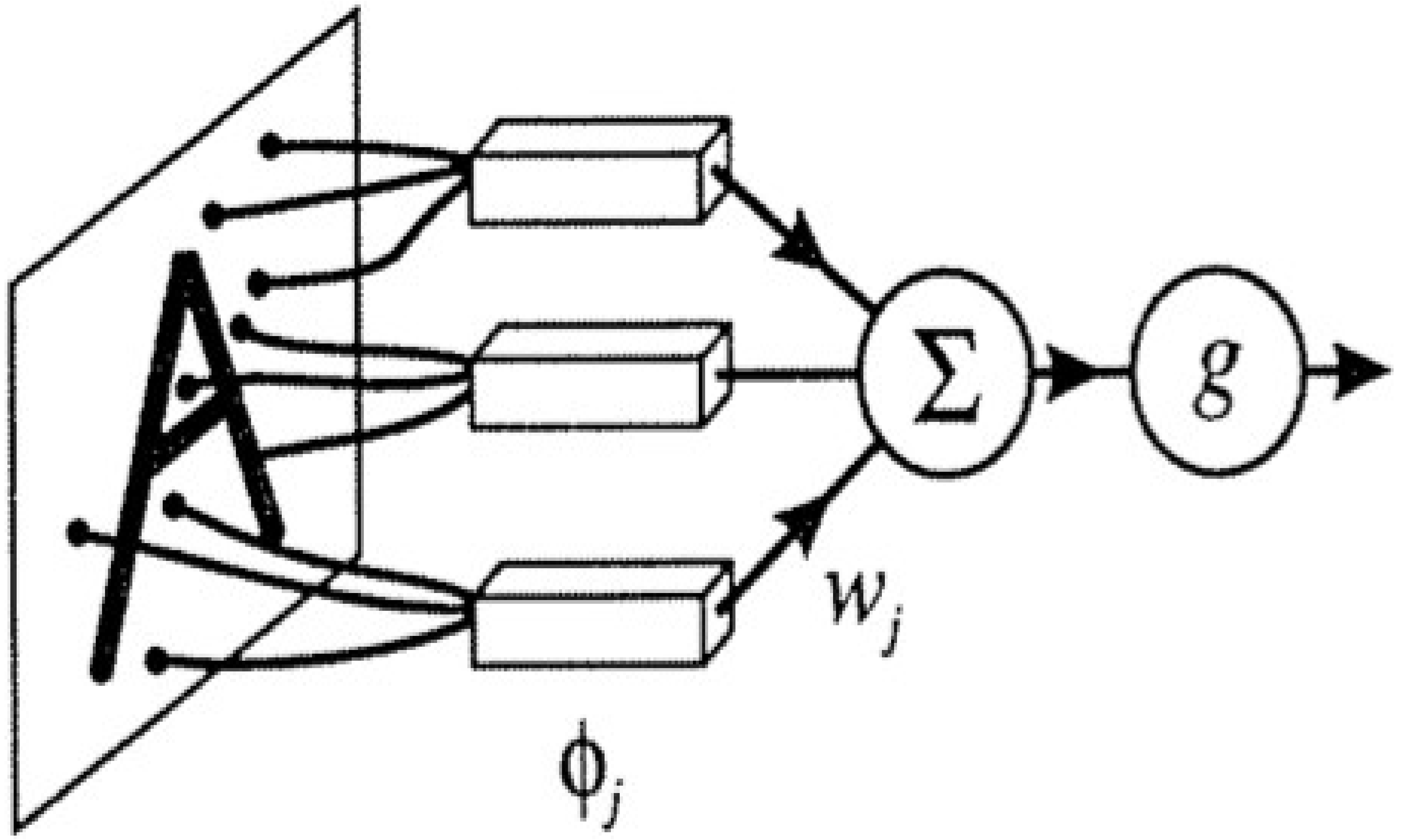
Psychological Review

THEODORE M. NEWCOMB, Editor
University of Michigan

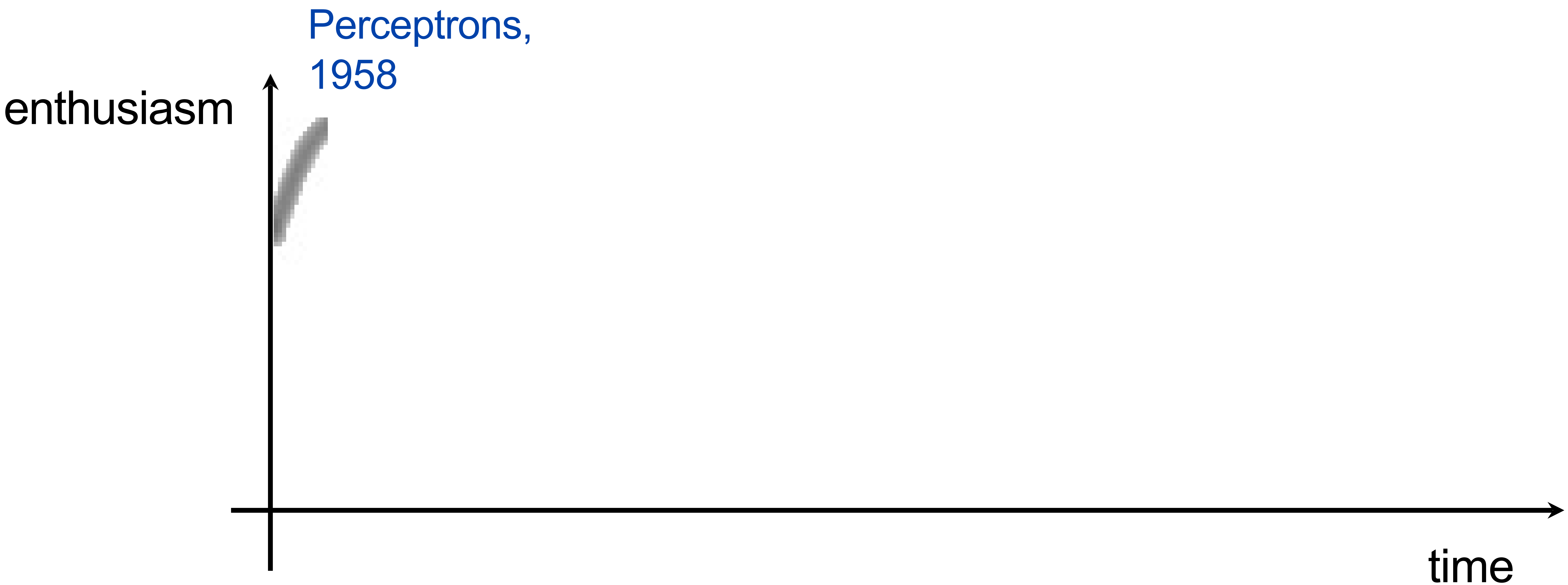
CONTENTS

This is the last issue of Volume 65. Title page and index for the volume appear herein.

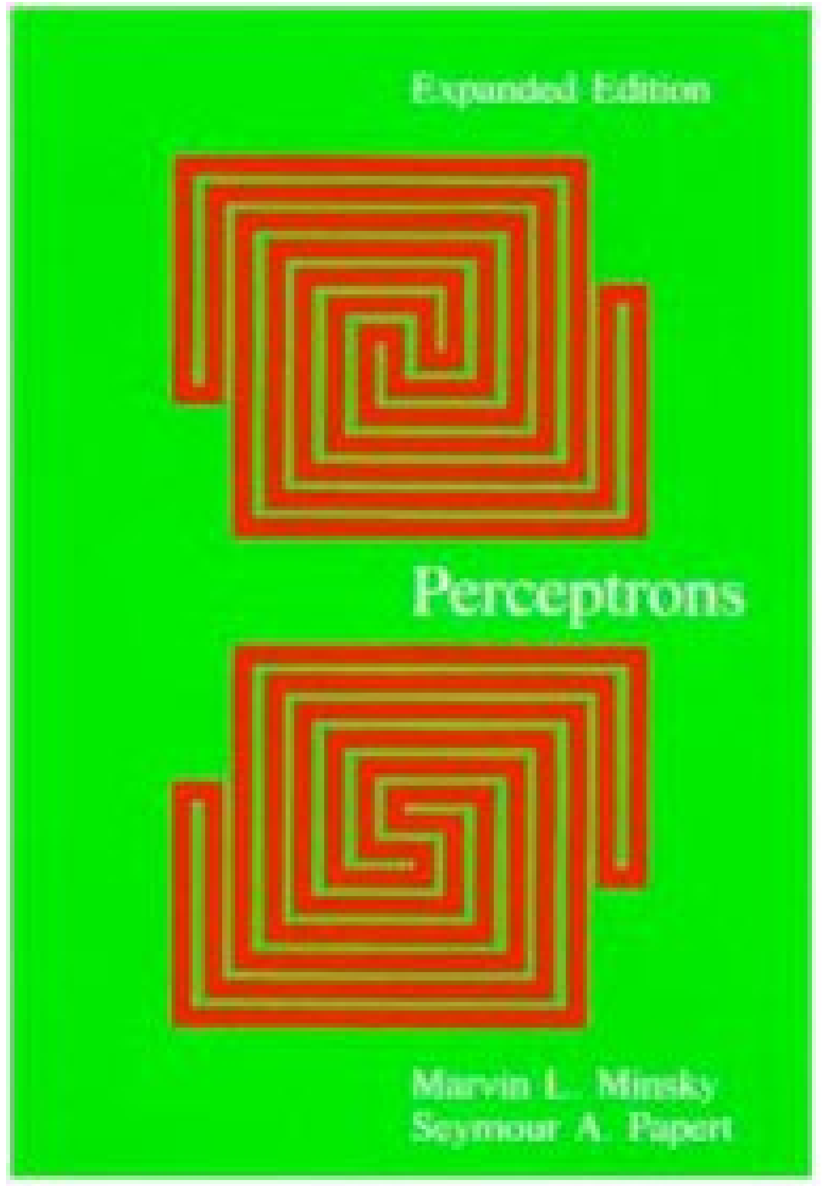
PUBLISHED BIMONTHLY BY THE
AMERICAN PSYCHOLOGICAL ASSOCIATION, INC.



http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf


Perceptrons,
1958
enthusiasm
time

Minsky and Papert, Perceptrons, 1972



Perceptrons, expanded edition

An Introduction to Computational Geometry

By Marvin Minsky and Seymour A. Papert

Overview

Perceptrons - the first systematic study of parallelism in computation - has remained a classical work on threshold automata networks for nearly two decades. It marked a historical turn in artificial intelligence, and it is required reading for anyone who wants to understand the connectionist counterrevolution that is going on today.

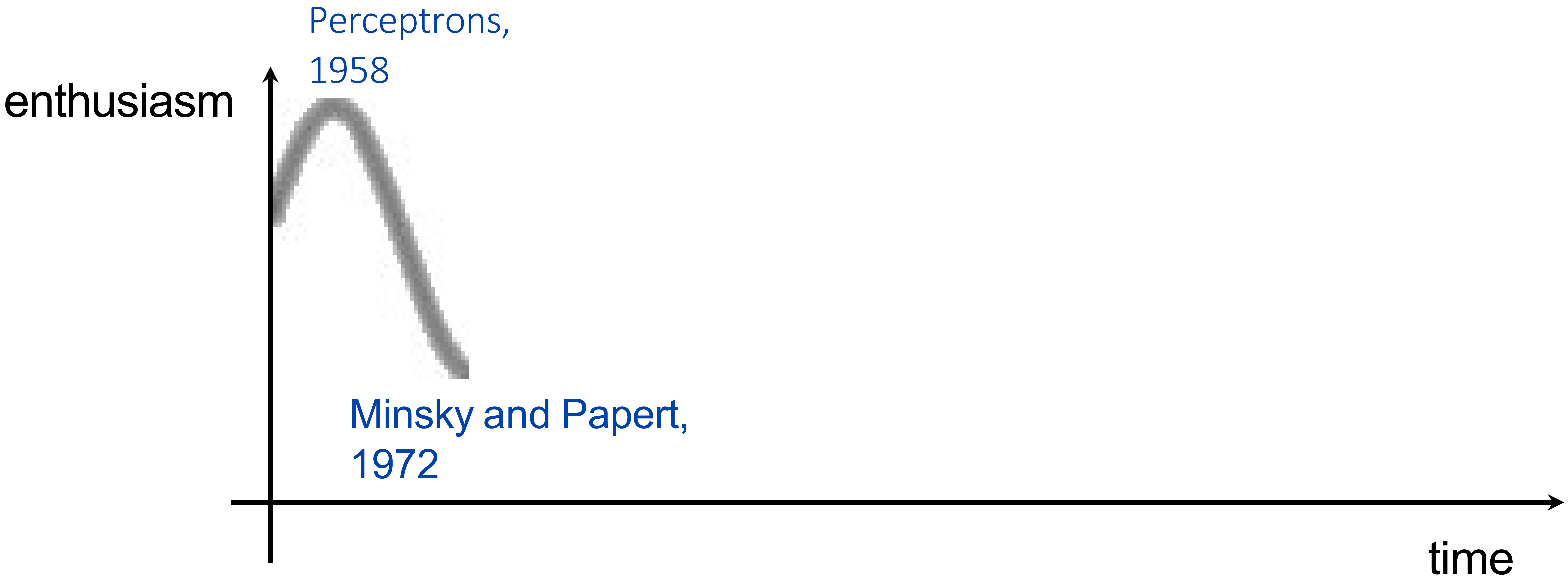
Artificial-intelligence research, which for a time concentrated on the programming of ton Neumann computers, is swinging back to the idea that intelligence might emerge from the activity of networks of neuronlike entities. Minsky and Papert's book was the first example of a mathematical analysis carried far enough to show the exact limitations of a class of computing machines that could seriously be considered as models of the brain. Now the new developments in mathematical tools, the recent interest of physicists in the theory of disordered matter, the new insights into and psychological models of how the brain works, and the evolution of fast computers that can simulate networks of automata have given *Perceptrons* new importance.

Witnessing the swing of the intellectual pendulum, Minsky and Papert have added a new chapter in which they discuss the current state of parallel computers, review developments since the appearance of the 1972 edition, and identify new research directions related to connectionism. They note a central theoretical challenge facing connectionism: the challenge to reach a deeper understanding of how "objects" or "agents" with individuality can emerge in a network. Progress in this area would link connectionism with what the authors have called "society theories of mind."

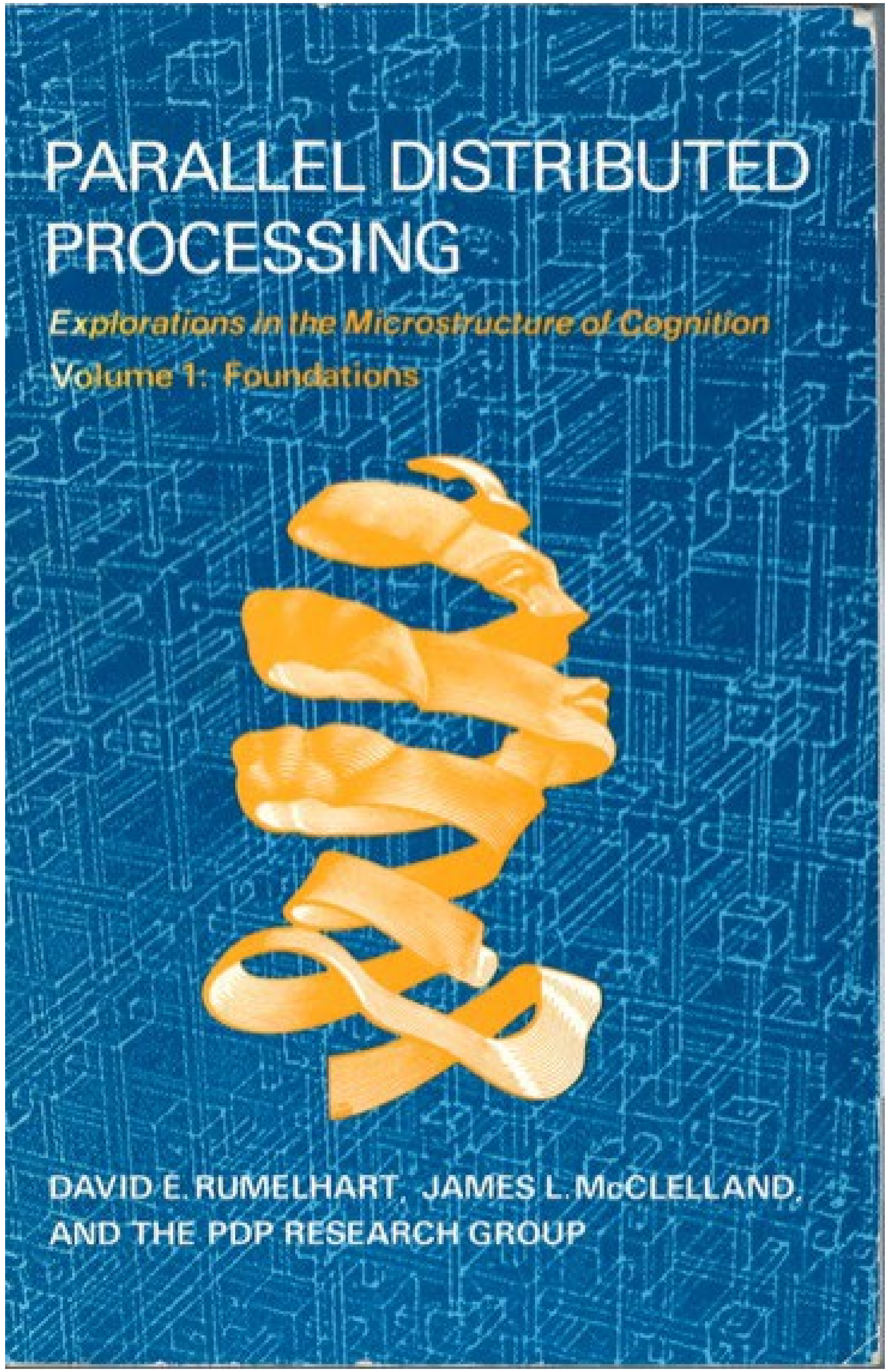
FOR BUYING OPTIONS, START HERE

Select Shipping Destination

Paperback | **$35.00 Short** | **£24.95** | ISBN: 9780262631112 | 308 pp. | 6 x 8.9 in | December 1987

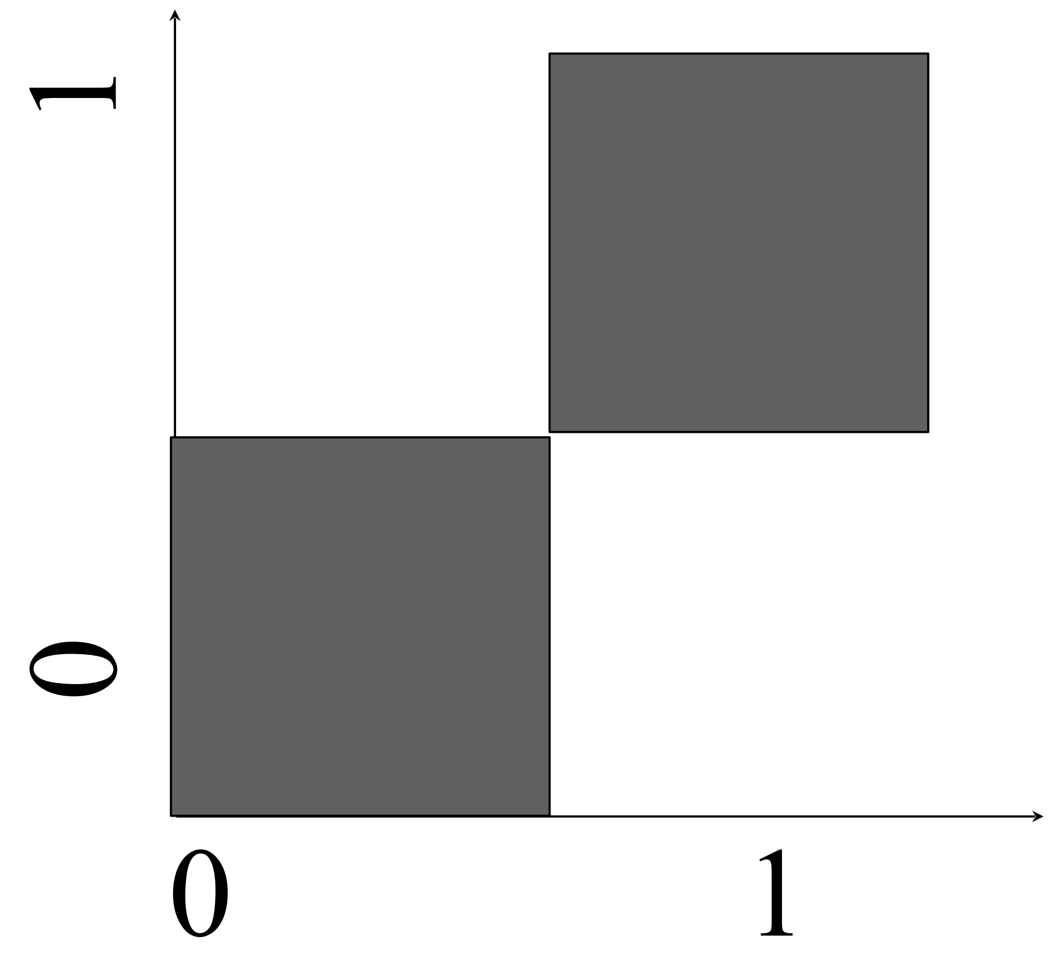

Perceptrons,
1958
enthusiasm
Minsky and Papert,
1972
time

Parallel Distributed Processing (PDP), 1986

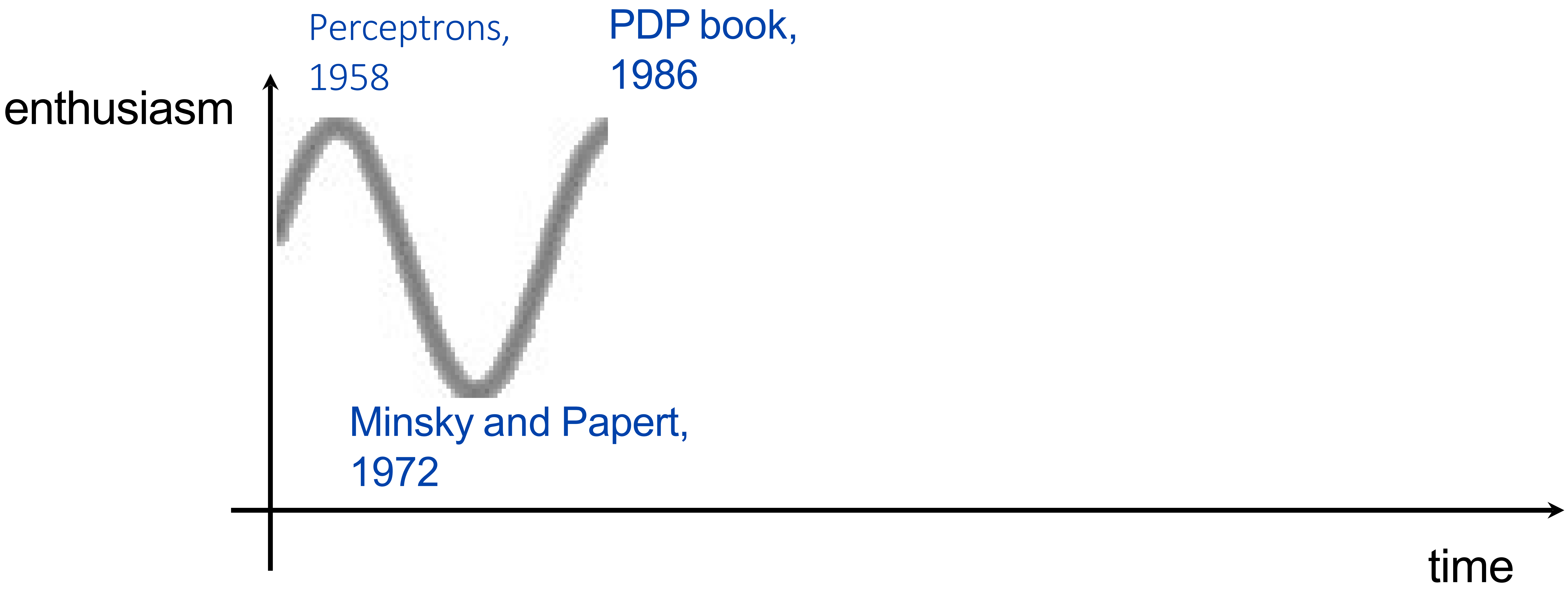


XOR problem

Inputs		Output
0	0	0
1	0	1
0	1	1
1	1	0



PDP authors pointed to the backpropagation algorithm as a breakthrough, allowing multi-layer neural networks to be trained. Among the functions that a multi-layer network can represent but a single-layer network cannot: the XOR function.



Perceptrons,
1958
PDP book,
1986
enthusiasm
Minsky and Papert,
1972
time

LeCun convolutional neural networks

PROC. OF THE IEEE, NOVEMBER 1998

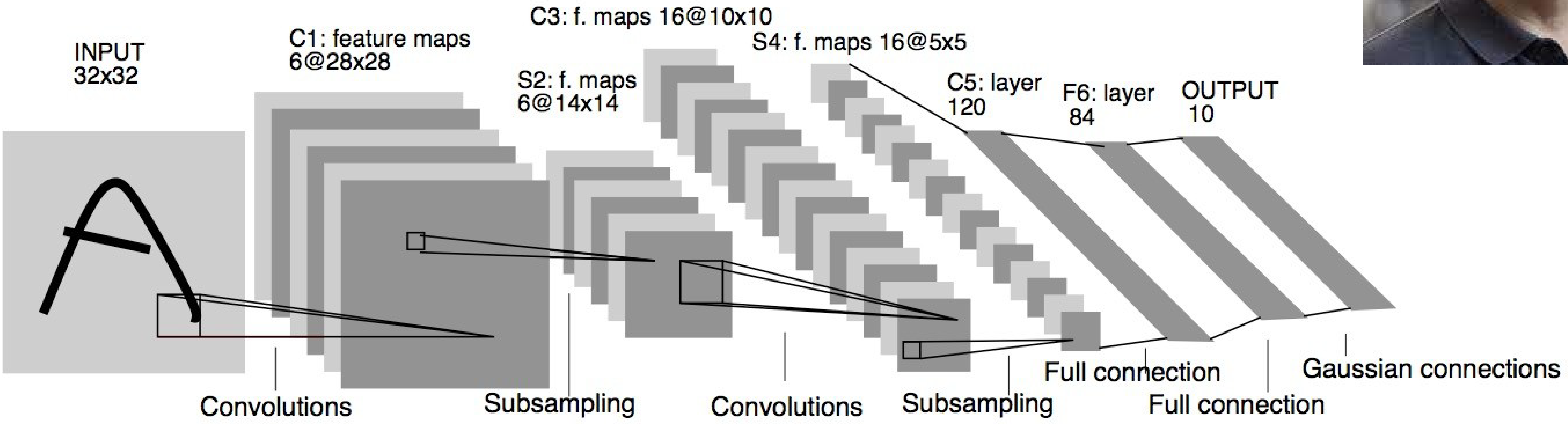


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Demos:
http://yann.lecun.com/exdb/lenet/index.html

Source: Isola, Torralba, Freeman

Yann LeCun

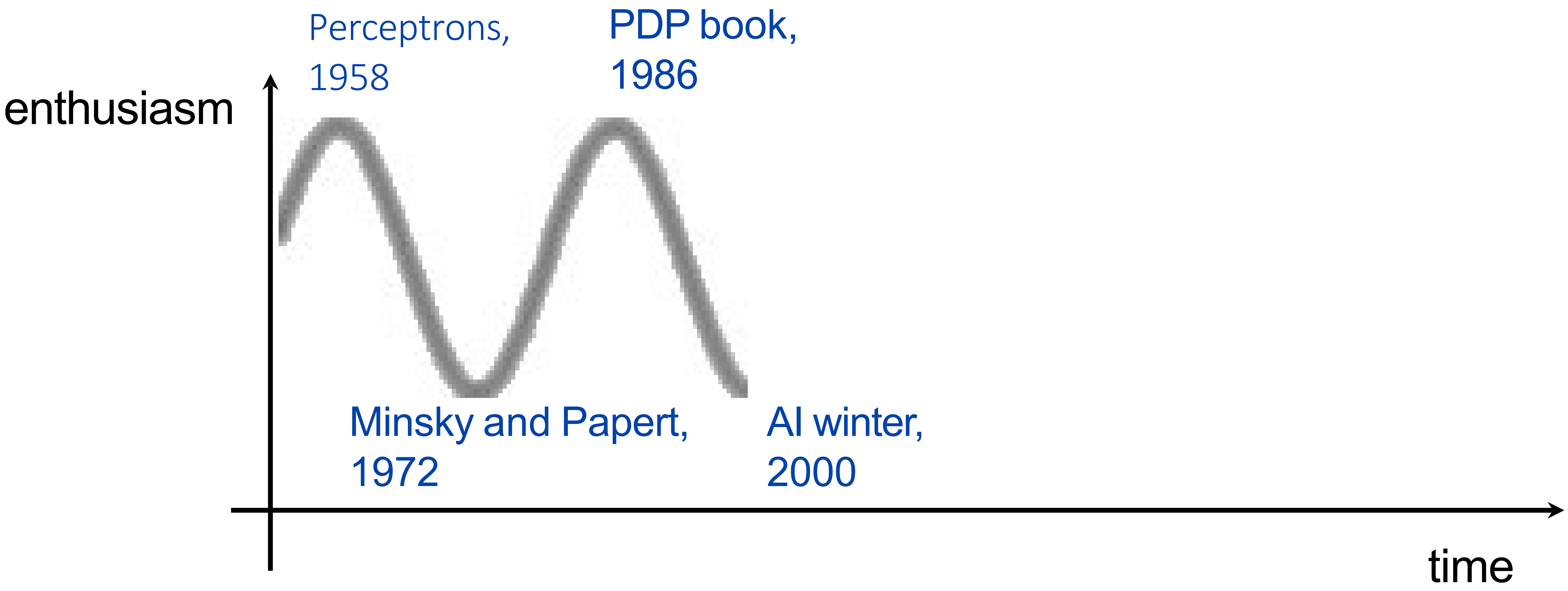
Was at Bell Labs when this video was recorded

Now
Prof @ NYU
Chief Scientist @ Meta

Turing Award 2018
(shared with Hinton and Bengio)

Neural Information Processing Systems 2000

- Neural Information Processing Systems is the premier conference on machine learning.
- Evolved from an interdisciplinary conference to a machine learning conference.
- For the 2000 conference:
  - title words predictive of paper acceptance: "Belief Propagation" and "Gaussian".
  - title words predictive of paper rejection: "Neural" and "Network".

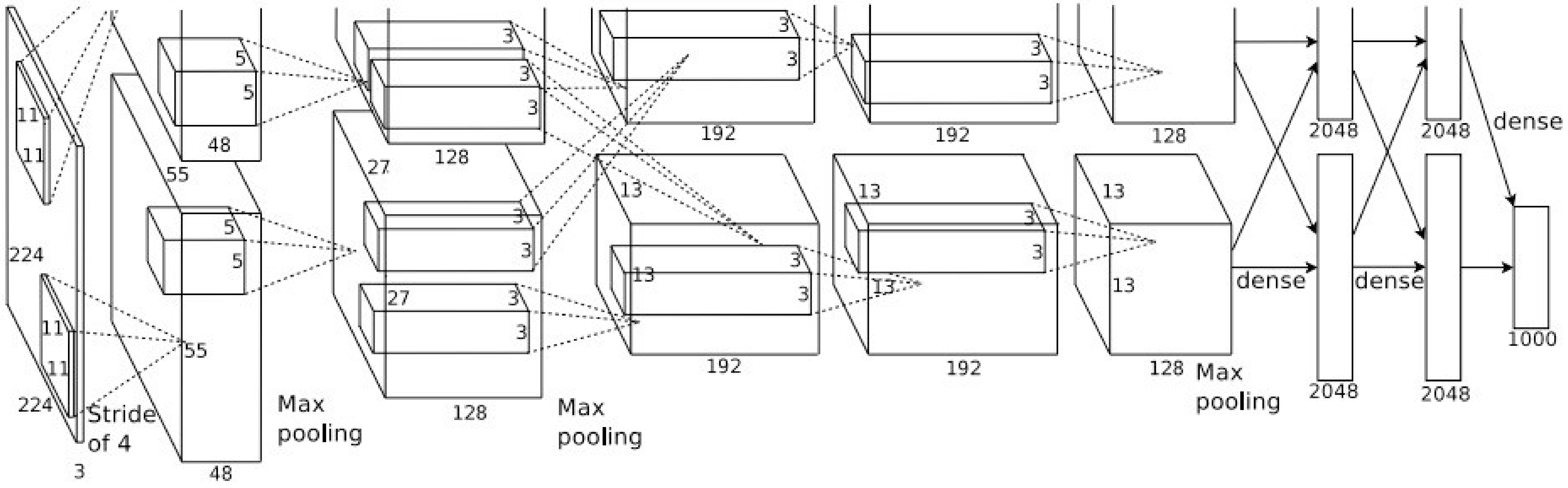

enthusiasm
Perceptrons, 1958
PDP book, 1986
Minsky and Papert, 1972
AI winter, 2000
time

ImageNet:
First (?) large-scale computer vision dataset

- Millions of images; 1000 categories
- **PI: Fei-Fei Li**
  - Then: Prof, Princeton
  - Now: Prof, Stanford
- 2019 Longuet-Higgins Prize
  - Some argued that Li deserved the 2018 Turing Award along with Hinton, LeCun, Bengio
  - Their work could not have been empirically tested without ImageNet!

Krizhevsky, Sutskever, and Hinton, NeurIPS 2012

"AlexNet"



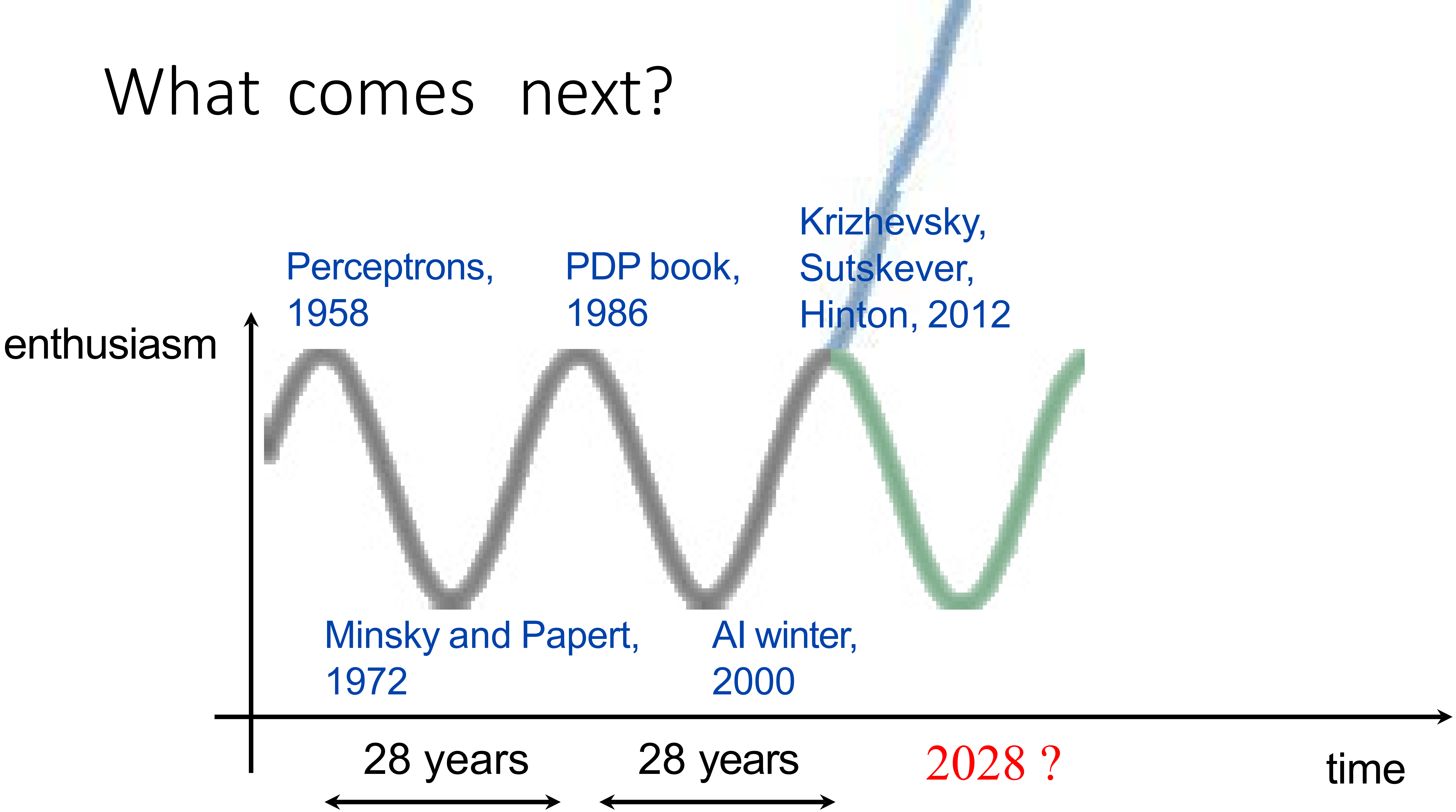
Got all the "pieces" right, e.g.,

- Trained on ImageNet
- 8 layer architecture (for reference: today we have architectures with 100+ layers)
- Allowed for multi-GP training

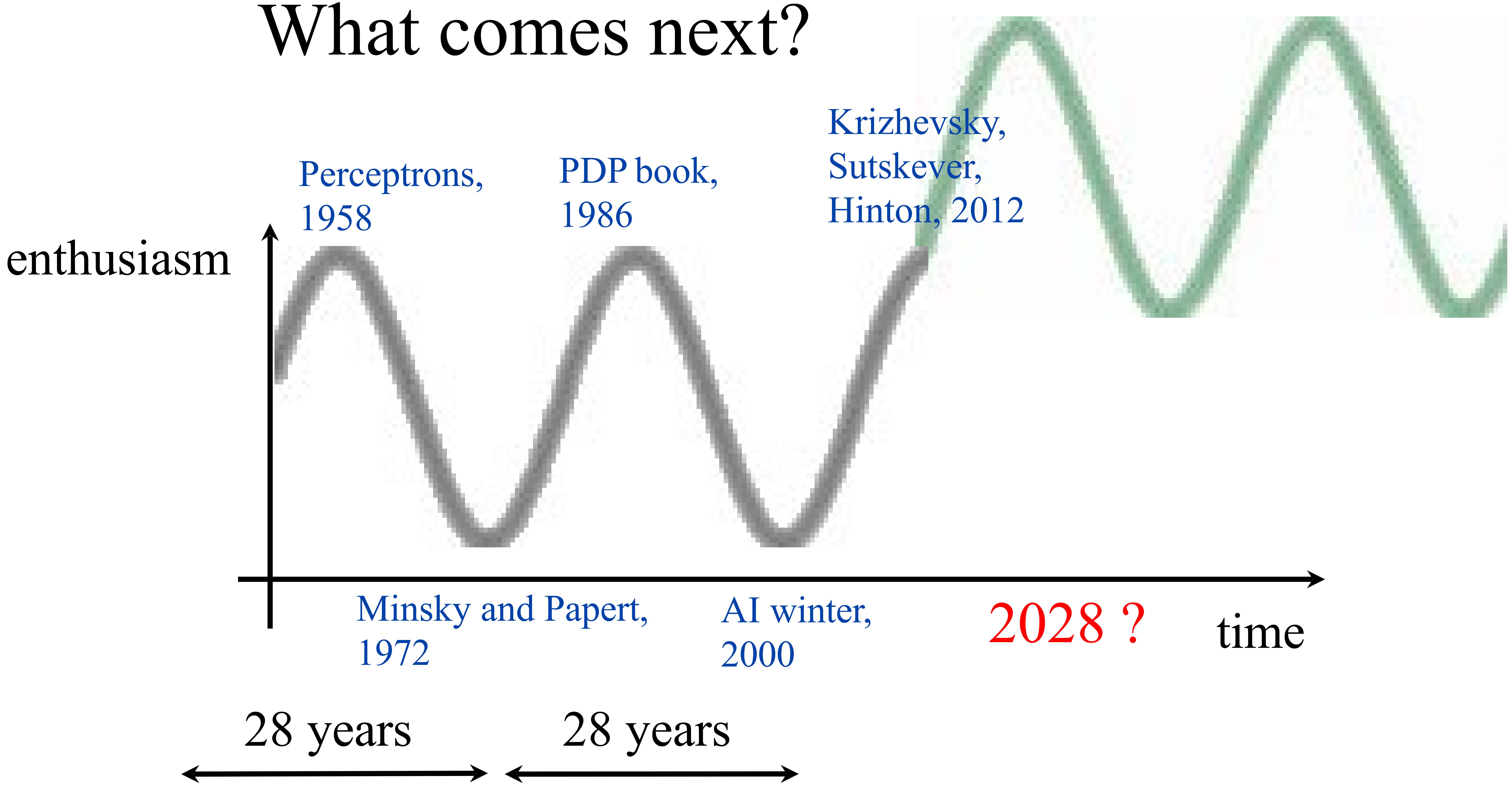
Krizhevsky, Sutskever, and Hinton, NeurIPS 2012


28 years
28 years
Perceptrons,
1958
PDP book,
1986
Krizhevsky,
Sutskever,
Hinton, 2012
enthusiasm
Minsky and Papert,
1972
AI winter,
2000
time

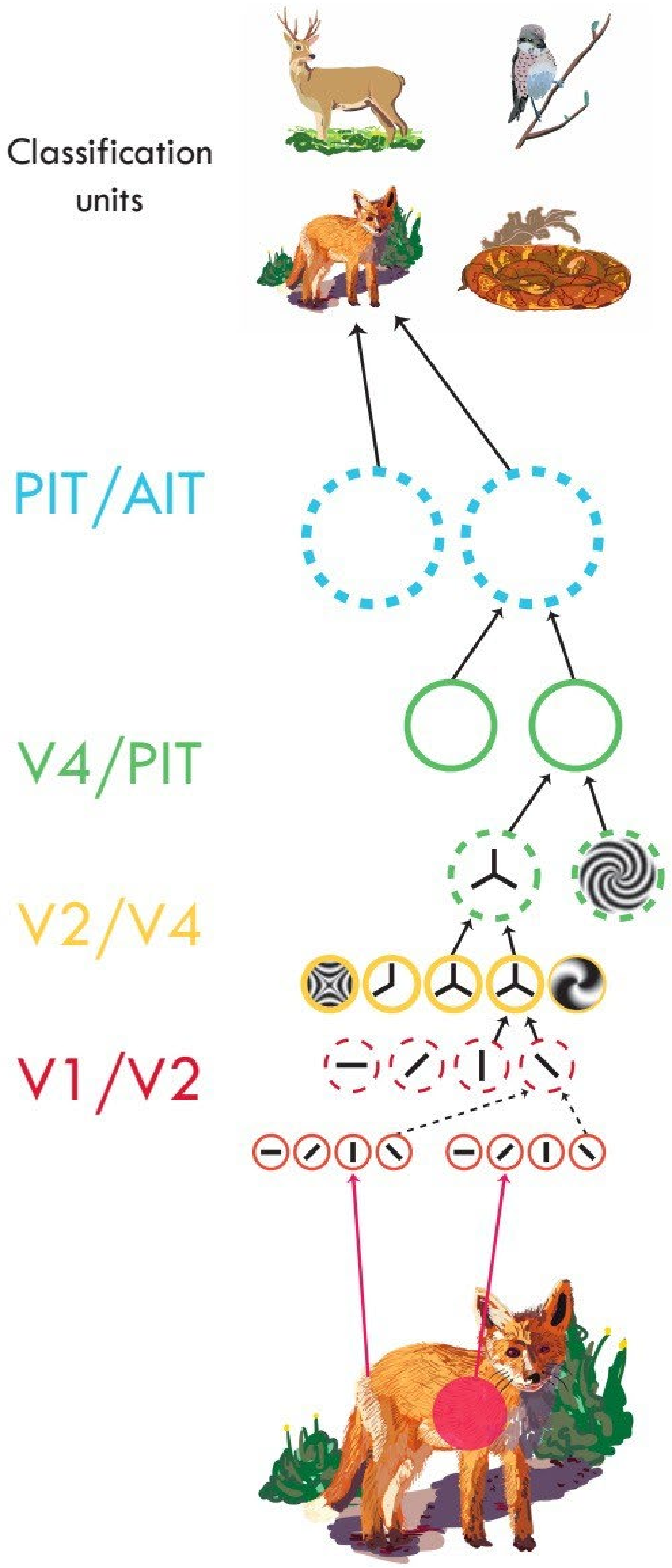
What comes next?



What comes next?



Inspiration: Hierarchical Representations



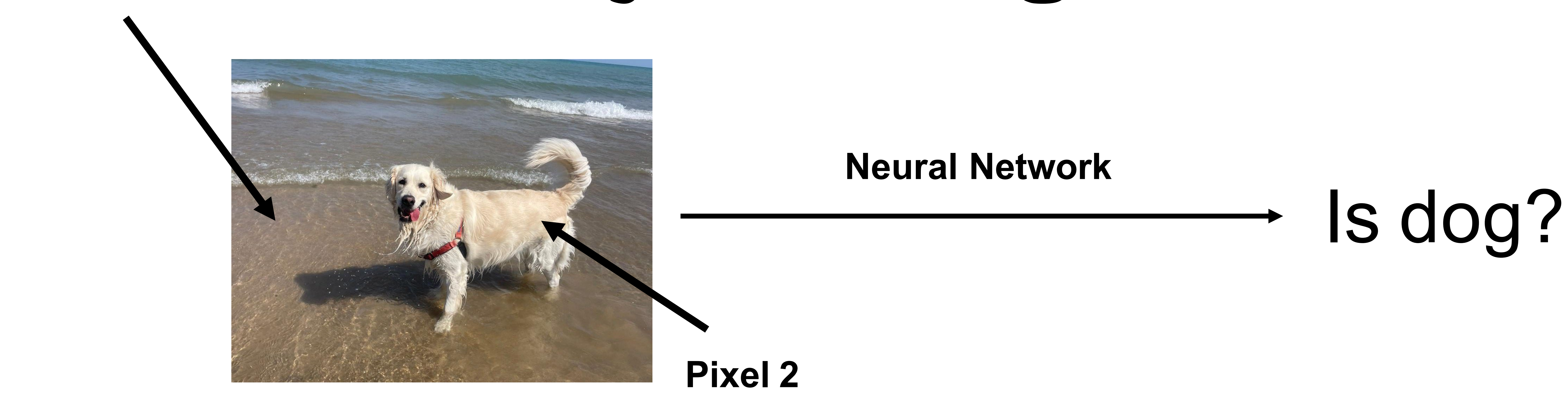
[Serre, 2014]

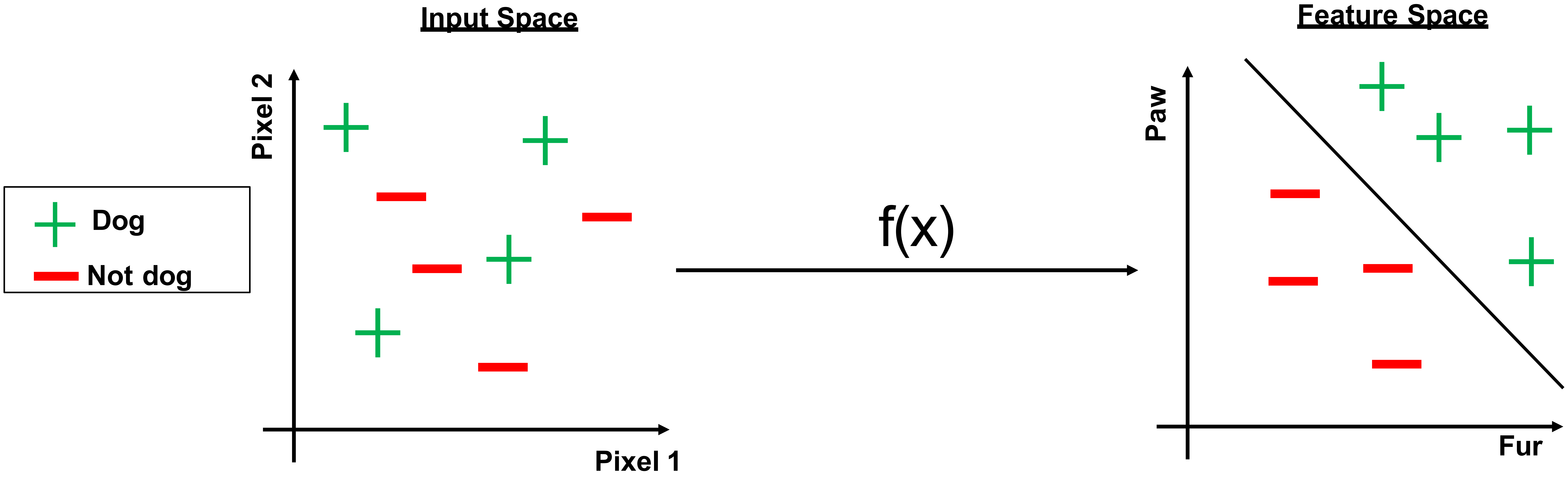
Best to treat as *inspiration*.

The neural nets we'll talk about aren't very biologically plausible.

Source: Isola, Torralba, Freeman

Object recognition





Goal: automatically learn a function that maps data from the input space to a feature space, i.e., "feature learning", rather than use hand-crafted features

Parametric Approach

Image

Array of **32x32x3** numbers
(3072 numbers total)

$f(\mathbf{x}, \mathbf{W})$

W

parameters
or weights

10 numbers giving
class scores

Parametric Approach: Linear Classifier

$$f(x,W) = Wx$$

Image

Array of **32x32x3** numbers
(3072 numbers total)

$f(\mathbf{x},\mathbf{W})$

10 numbers giving
class scores

W

parameters
or weights

Parametric Approach: Linear Classifier

3072x1

$$\boxed{f(x,W)} = \boxed{W}\boxed{x}$$

10x1 10x3072

Image

Array of **32x32x3** numbers
(3072 numbers total)

$\longrightarrow$ $f(\mathbf{x},\mathbf{W})$ $\longrightarrow$ **10** numbers giving class scores

$\uparrow$

W

parameters
or weights

Parametric Approach: Linear Classifier

$$\underset{\text{10x1}}{\boxed{f(x,W)}} = \underset{\text{10x3072}}{\boxed{W}}\overset{\text{3072x1}}{\boxed{x}} + \boxed{b}\quad \text{10x1}$$

Image

Array of **32x32x3** numbers
(3072 numbers total)

$$\longrightarrow f(\mathbf{x},\mathbf{W}) \longrightarrow$$

10 numbers giving
class scores

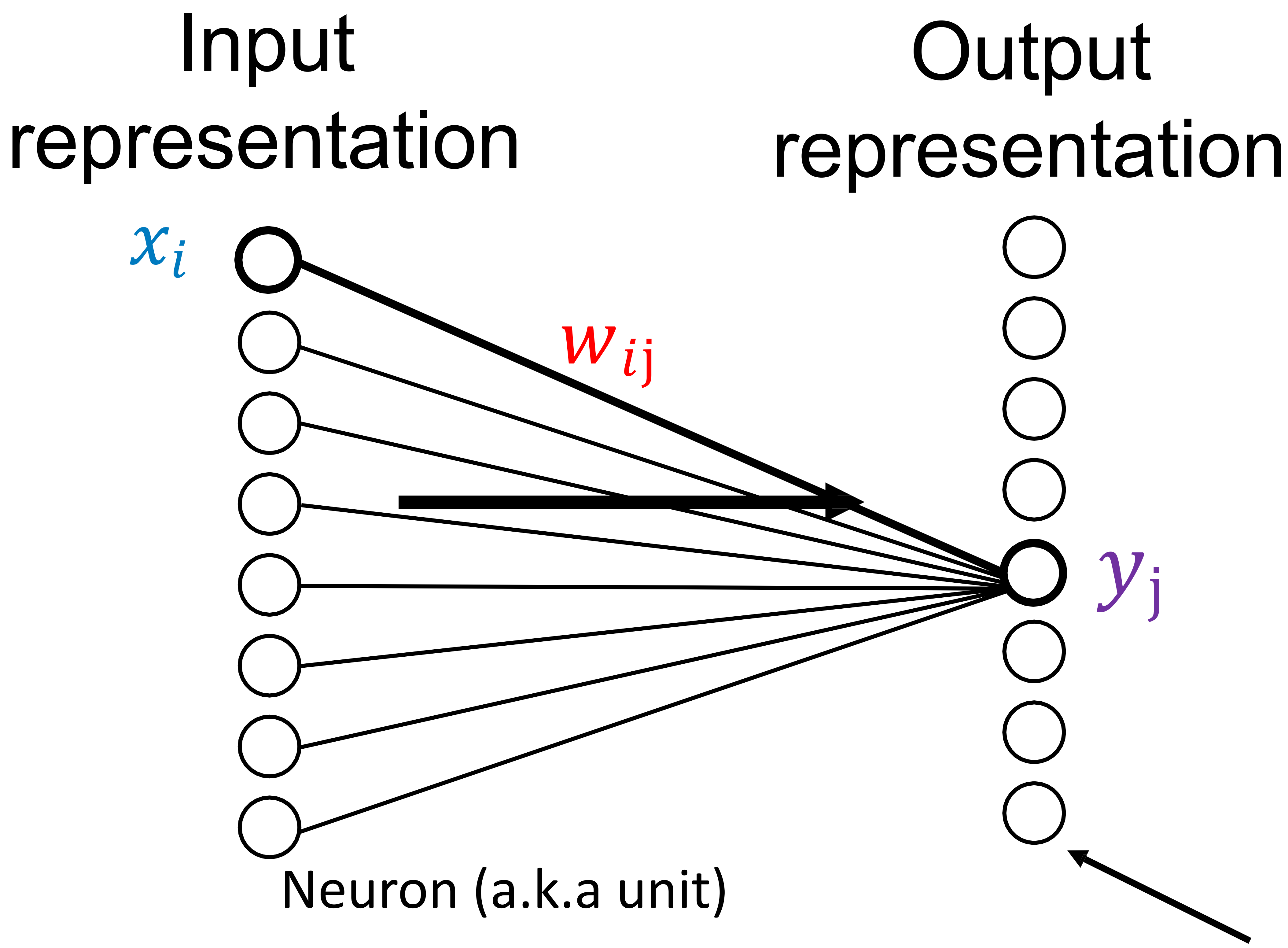
W

parameters
or weights

Computation in a neural net

Let's say we have some 1D input that we want to convert to some new feature space:

<u>Linear layer</u>



weights

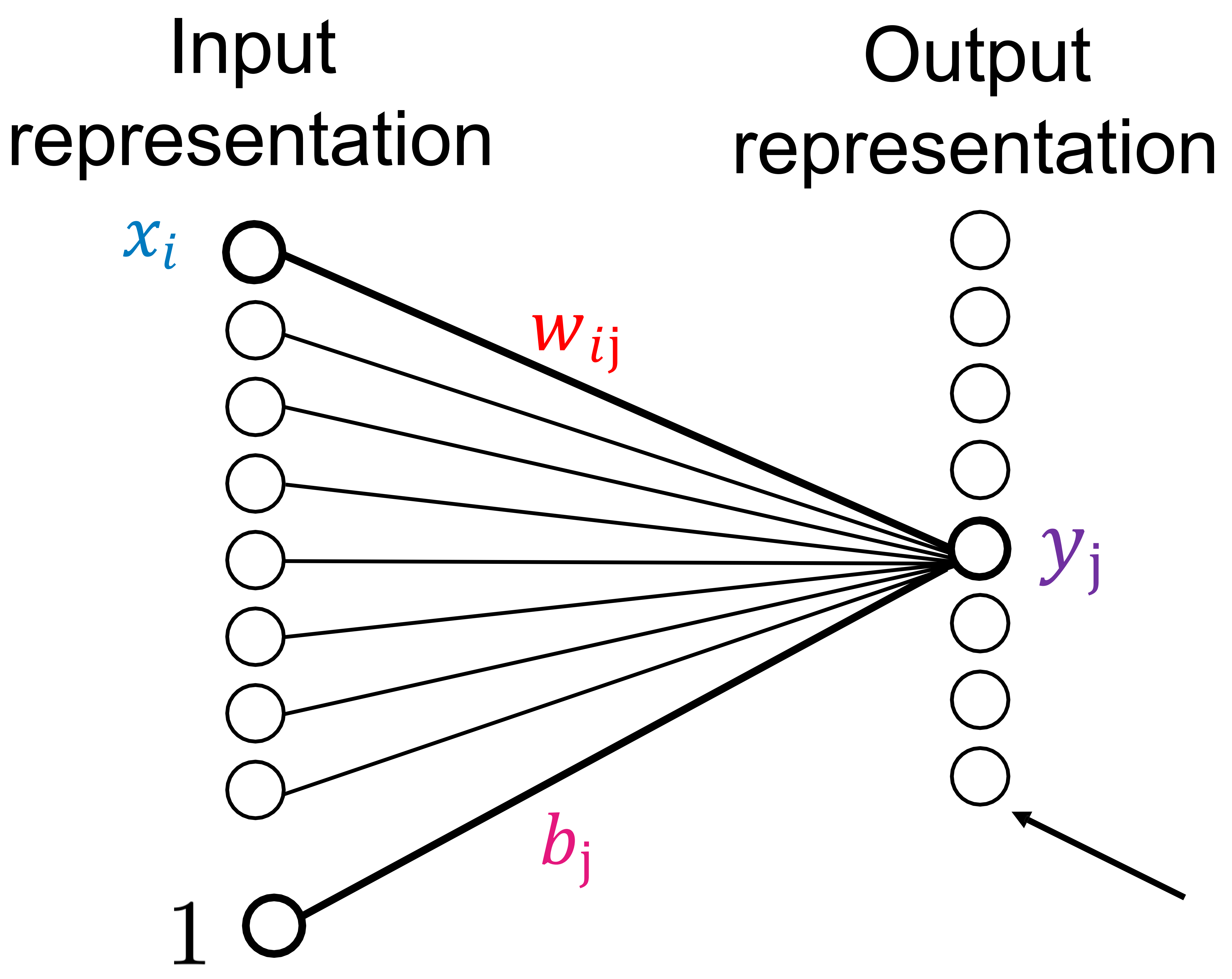
$$y_j = \sum_i w_{ij} x_i$$

Adapted from: Isola, Torralba, Freeman

Computation in a neural net

Let's say we have some 1D input that we want to convert to some new feature space

Linear layer



$$y_j = \sum_i w_{ij} x_i + b_j$$

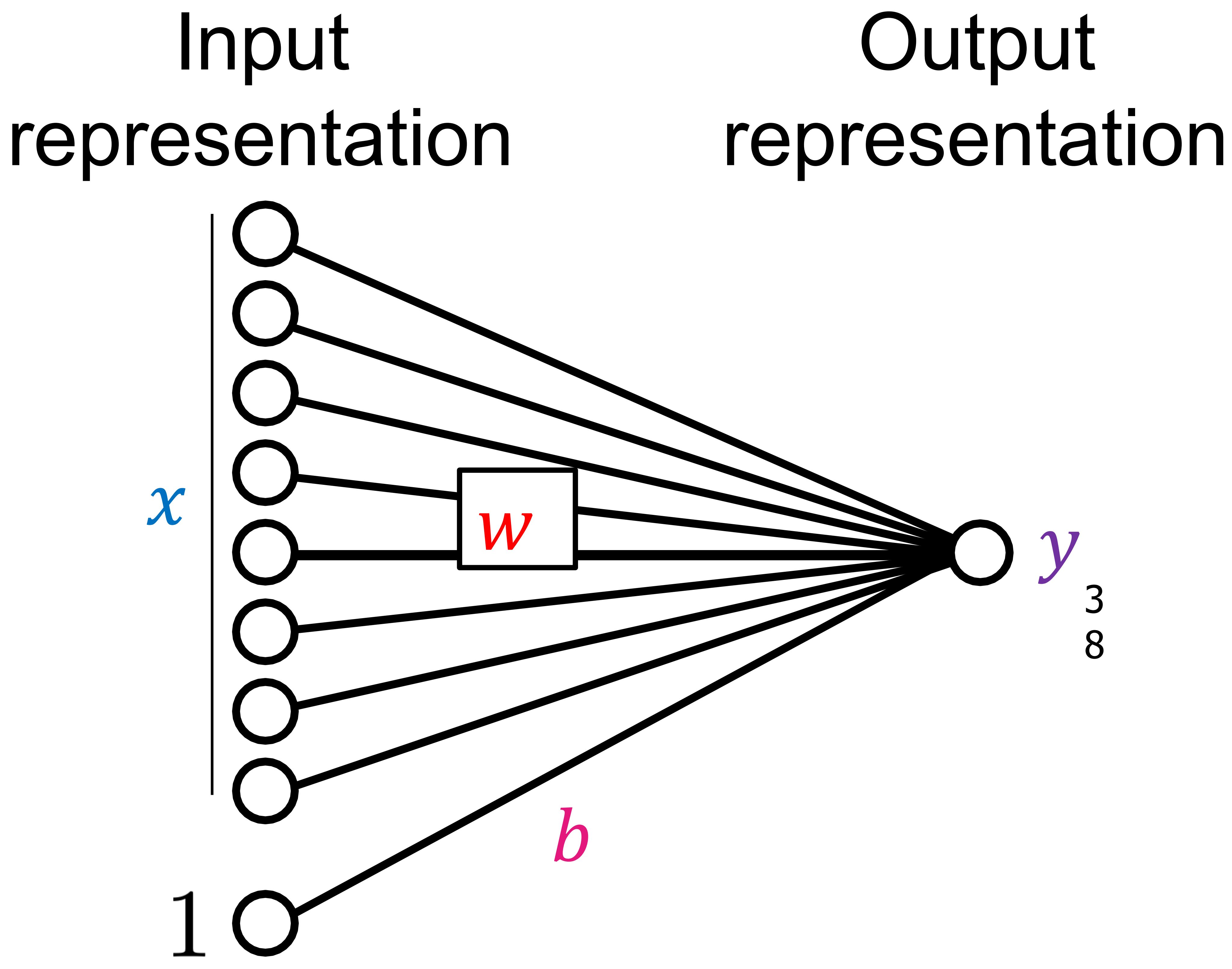
weights

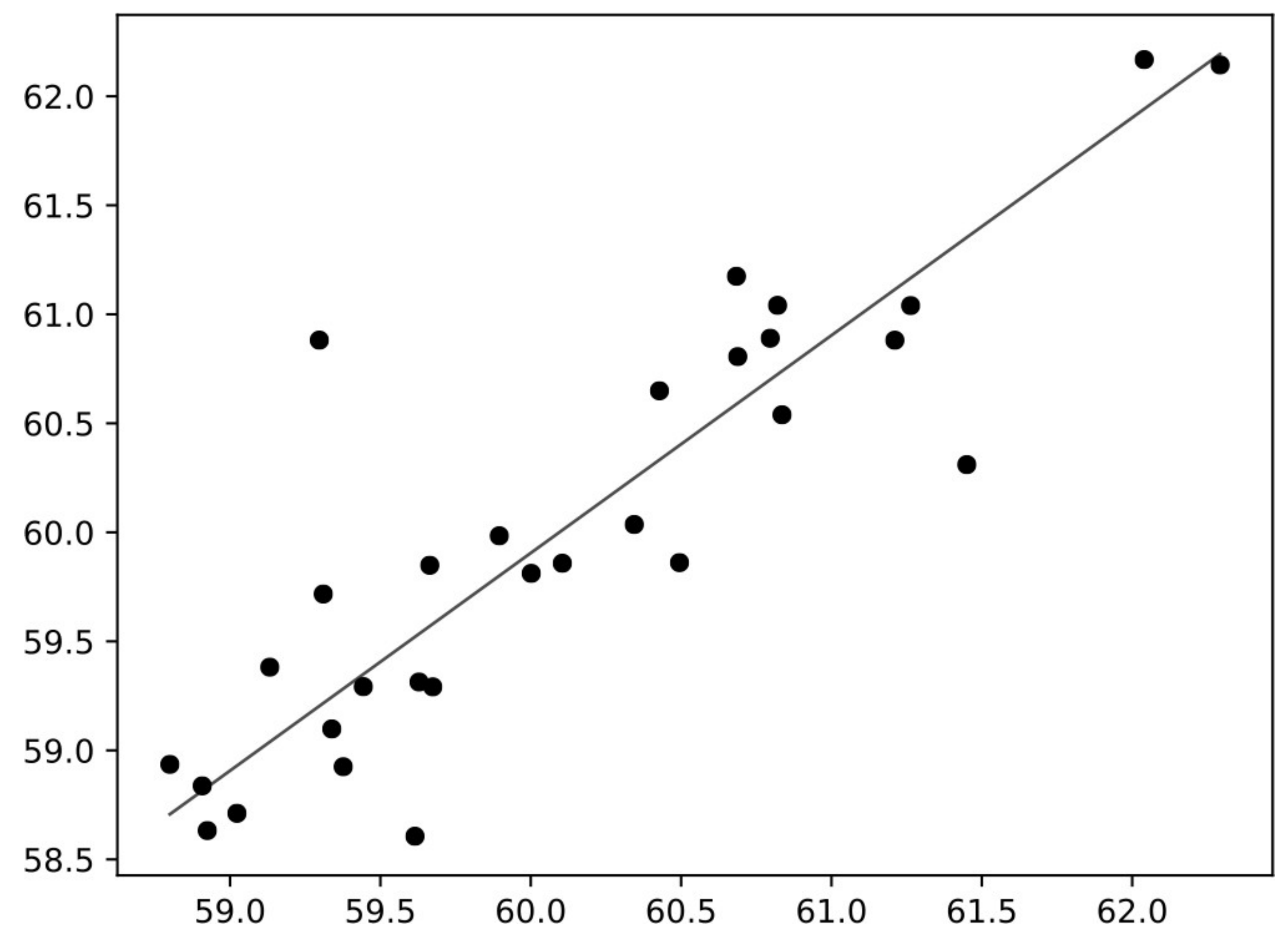
bias

Adapted from: Isola, Torralba, Freeman

Example: Linear Regression

Linear layer



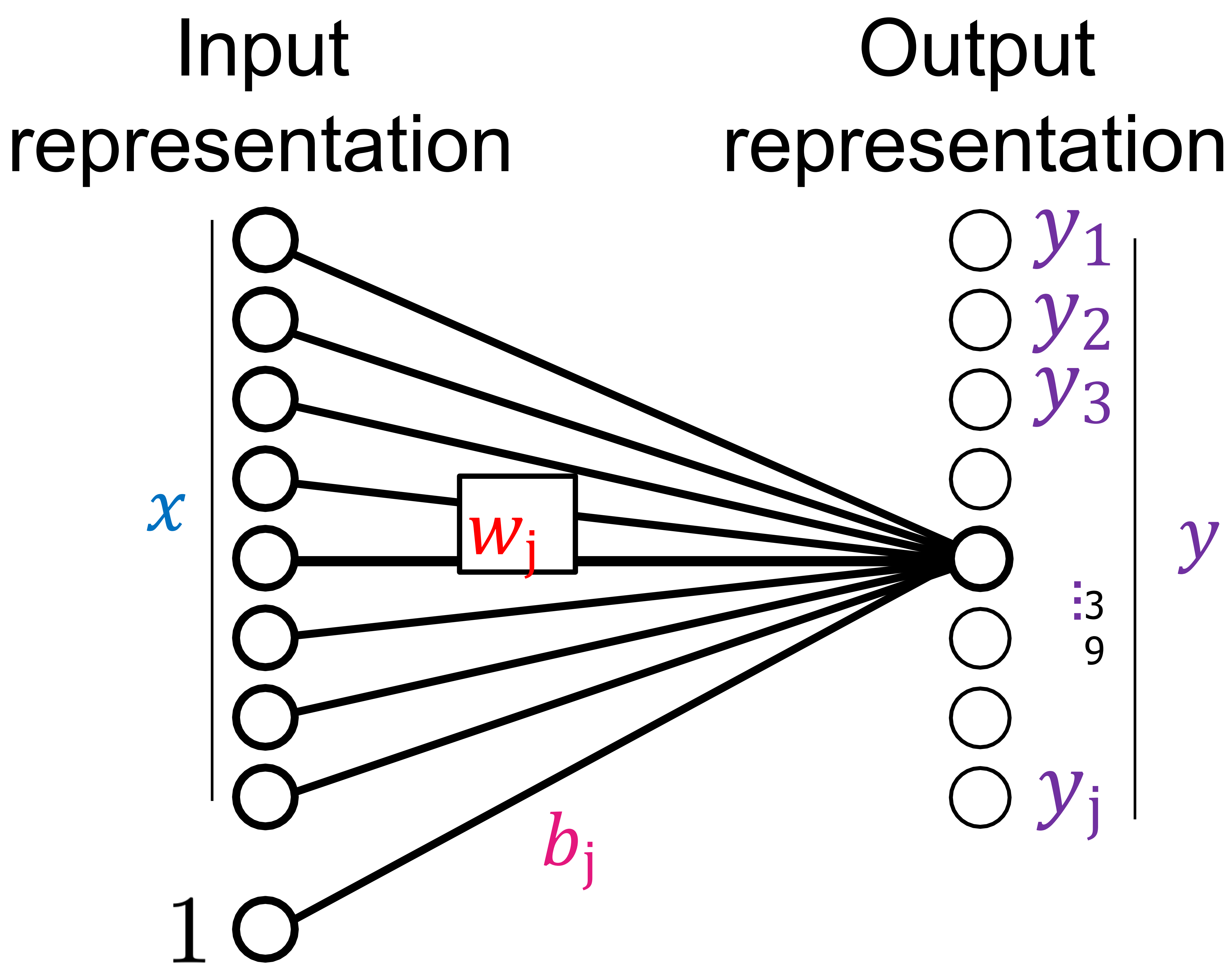


$$f_{\mathbf{w},b}(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + b$$

Adapted from: Isola, Torralba, Freeman

Computation in a neural net – Full Layer

Linear layer



$$y = Wx + b$$

$$\begin{bmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{j1} & \cdots & w_{jn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_j \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_j \end{bmatrix}$$

parameters of the model: $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{b}\}$

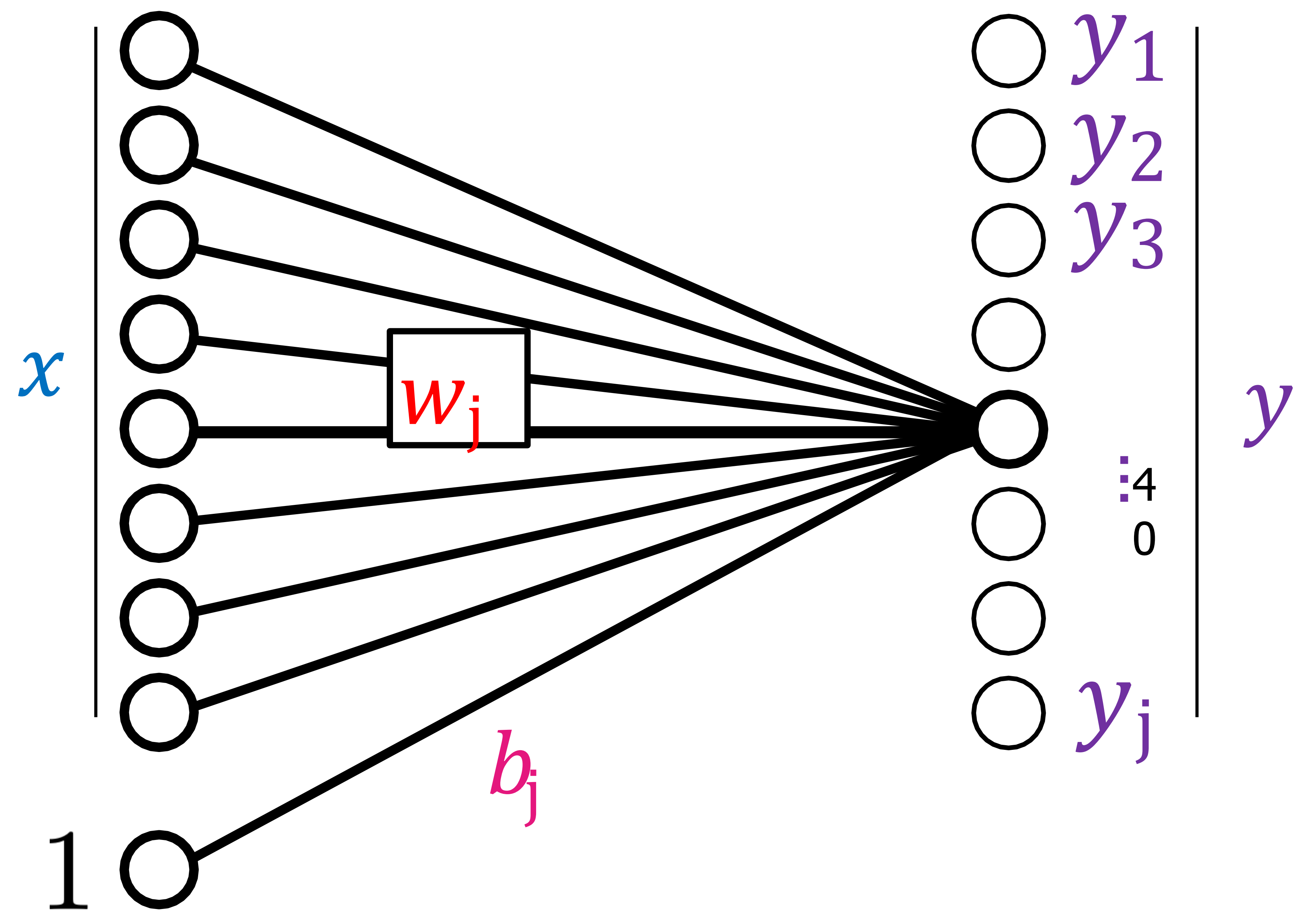
Adapted from: Isola, Torralba, Freeman

Computation in a neural net – Full Layer

Linear layer

Input representation

Output representation



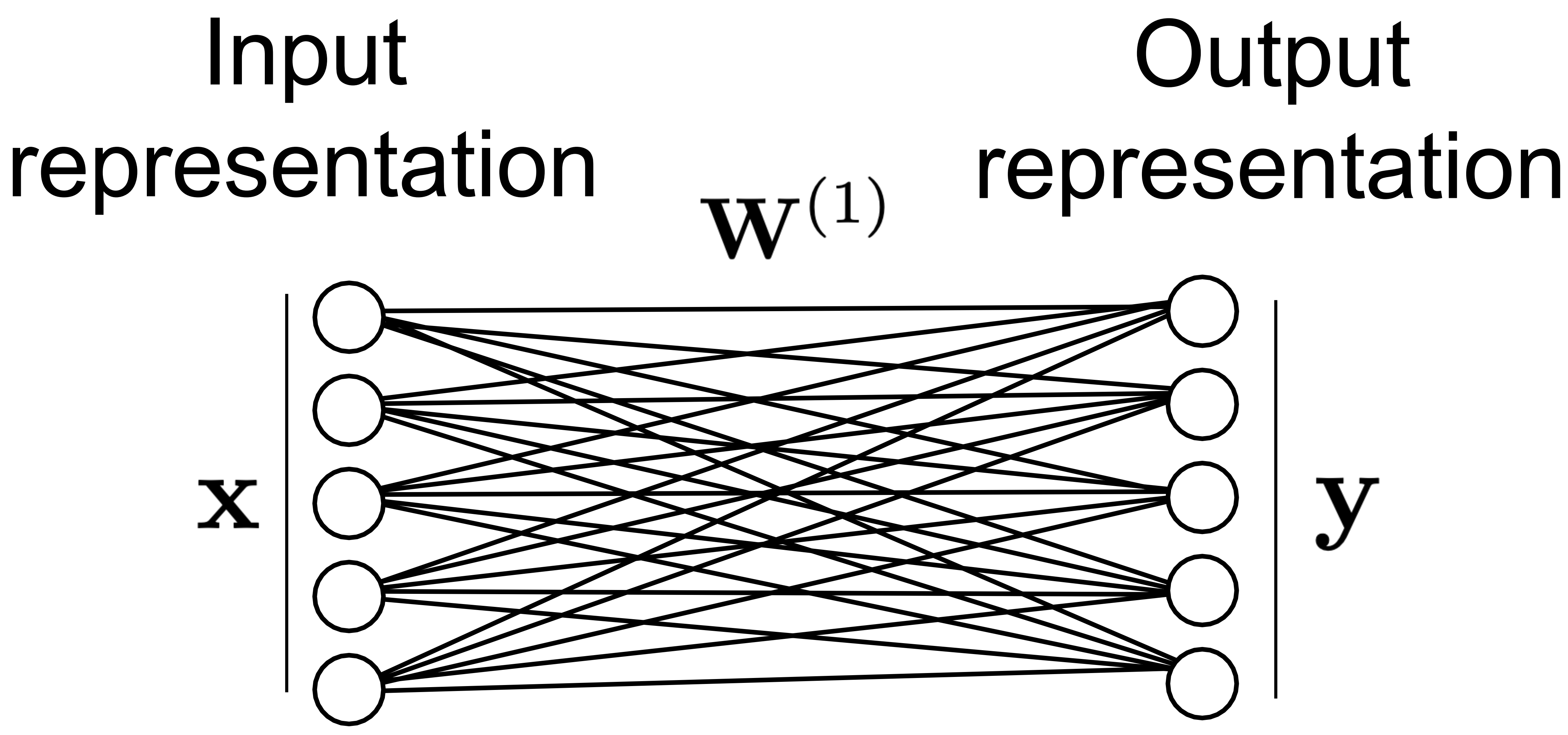
Full layer

$$y = Wx + b$$

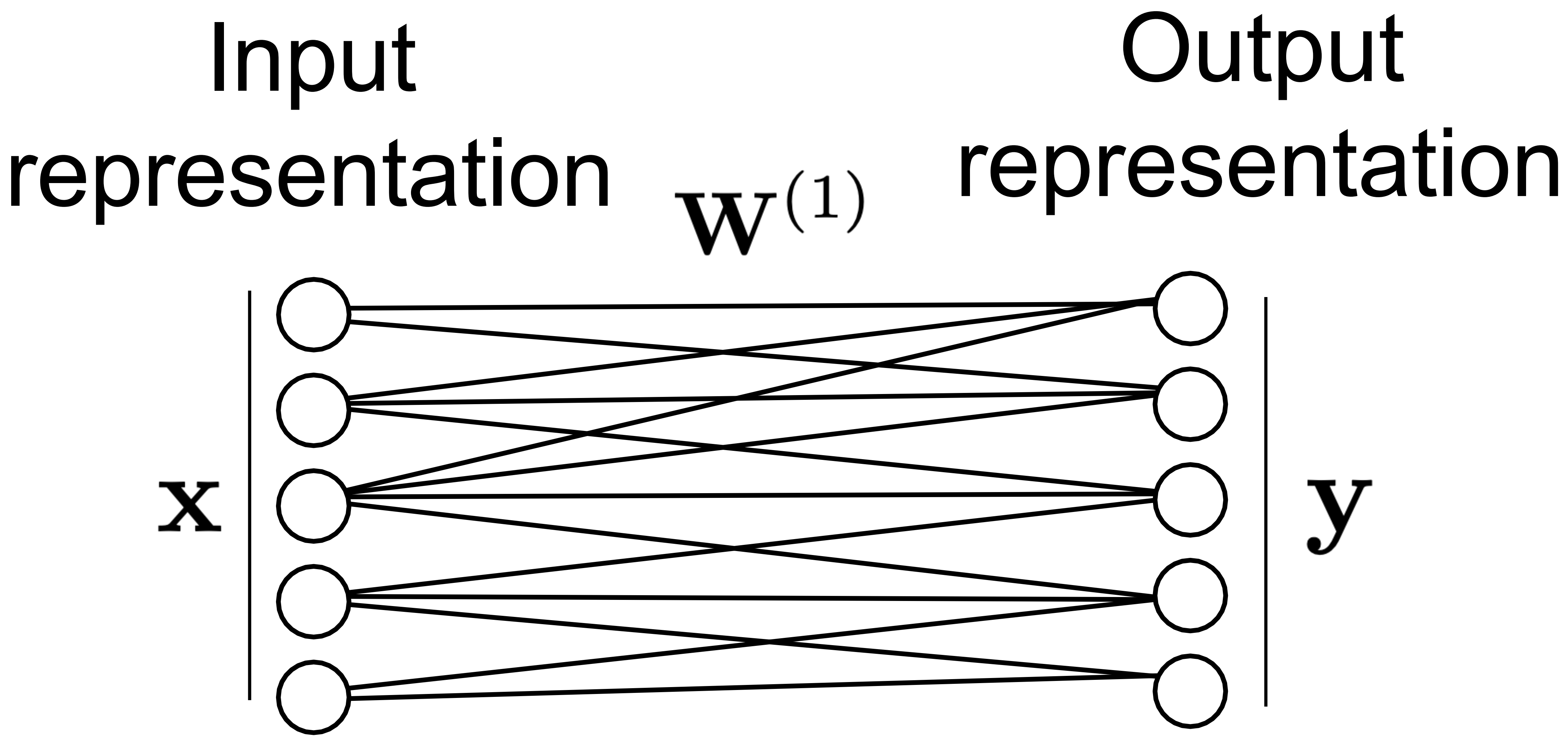
$$\begin{bmatrix} w_{11} & \cdots & w_{jn} & b_1 \\ \vdots & \ddots & \vdots & \vdots \\ w_{j1} & \cdots & w_{jn} & b_j \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \\ 1 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_j \end{bmatrix}$$

Can again simplify notation by appending a 1 to $\mathbf{x}$

Connectivity patterns



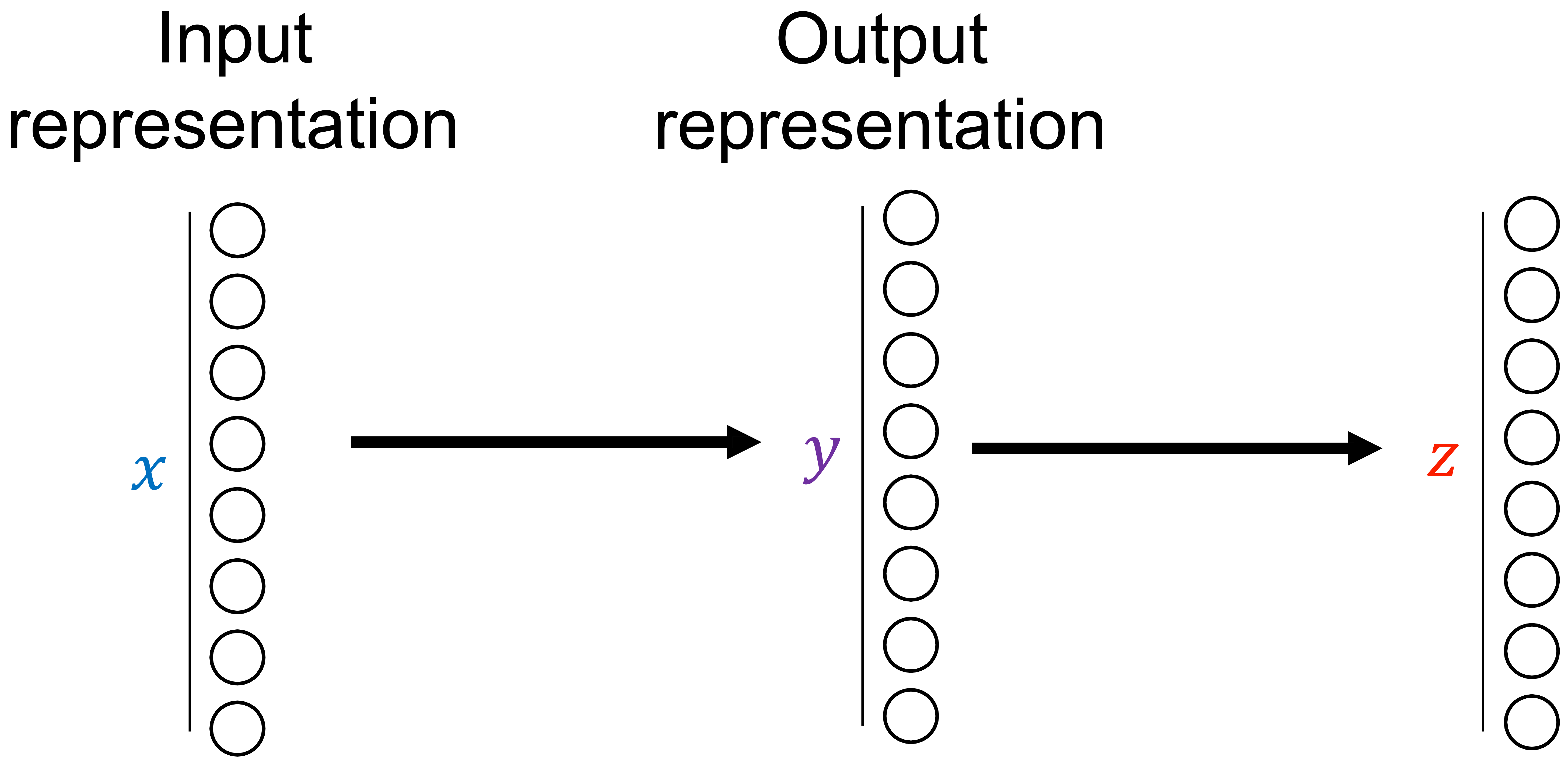
Fully connected layer



Locally connected layer
(Sparse W)

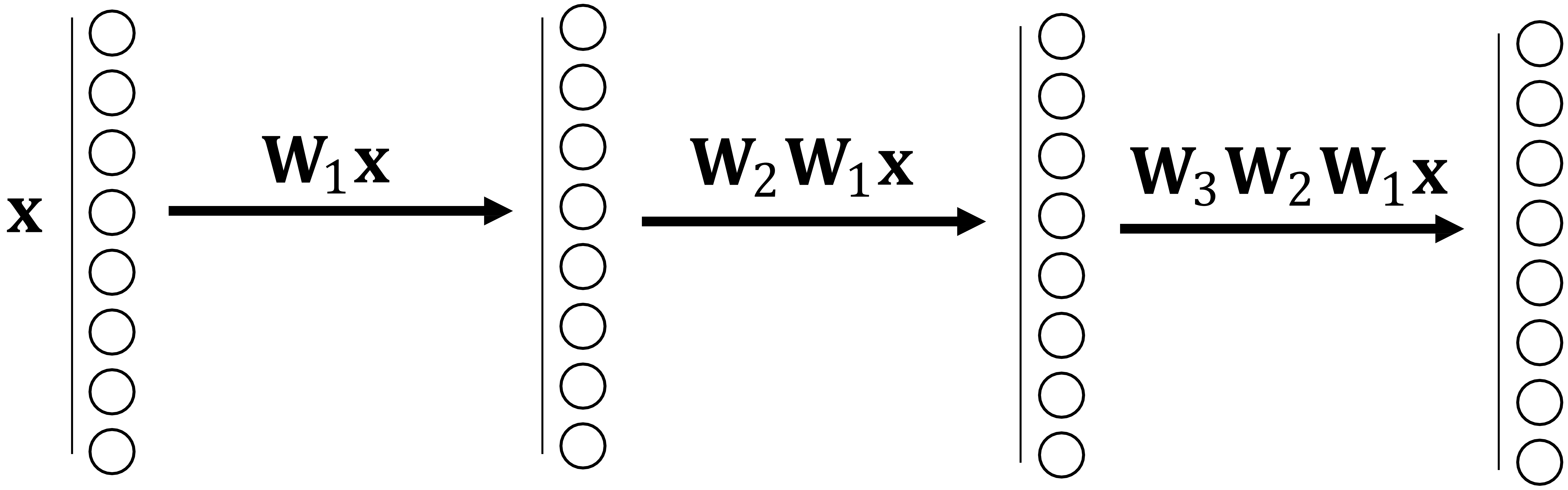
Computation in a neural net – Recap

We can now transform our input representation vector into some output representation vector using a bunch of linear combinations of the input:

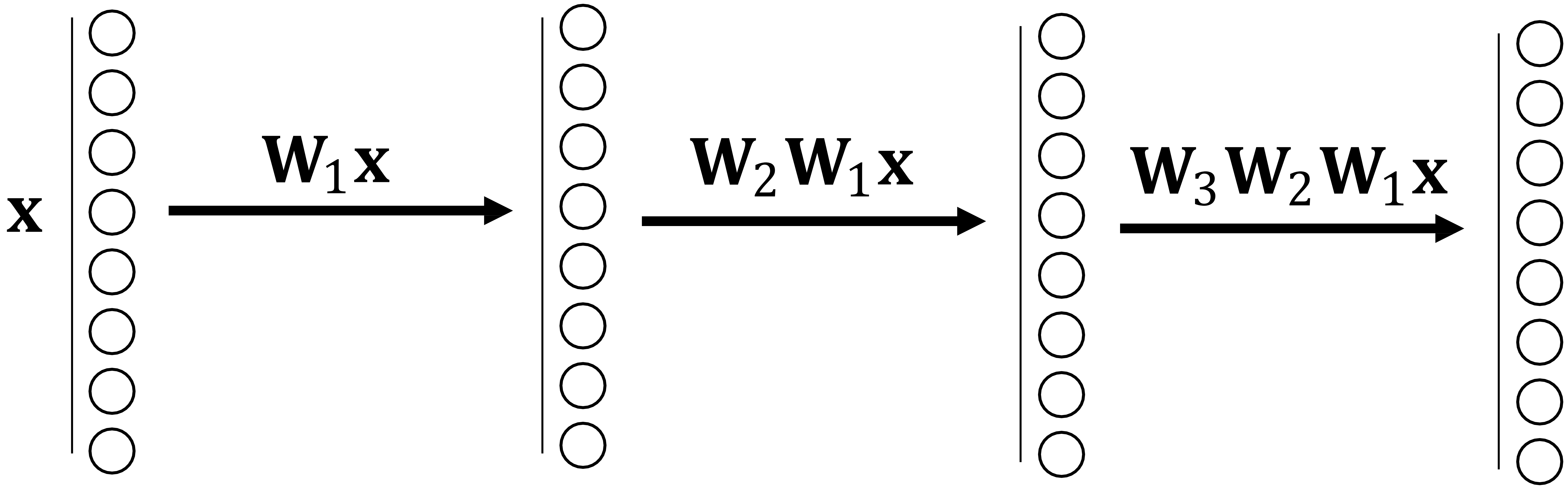


We can repeat this as many times as we want!

What is the problem with this idea?



What is the problem with this idea?

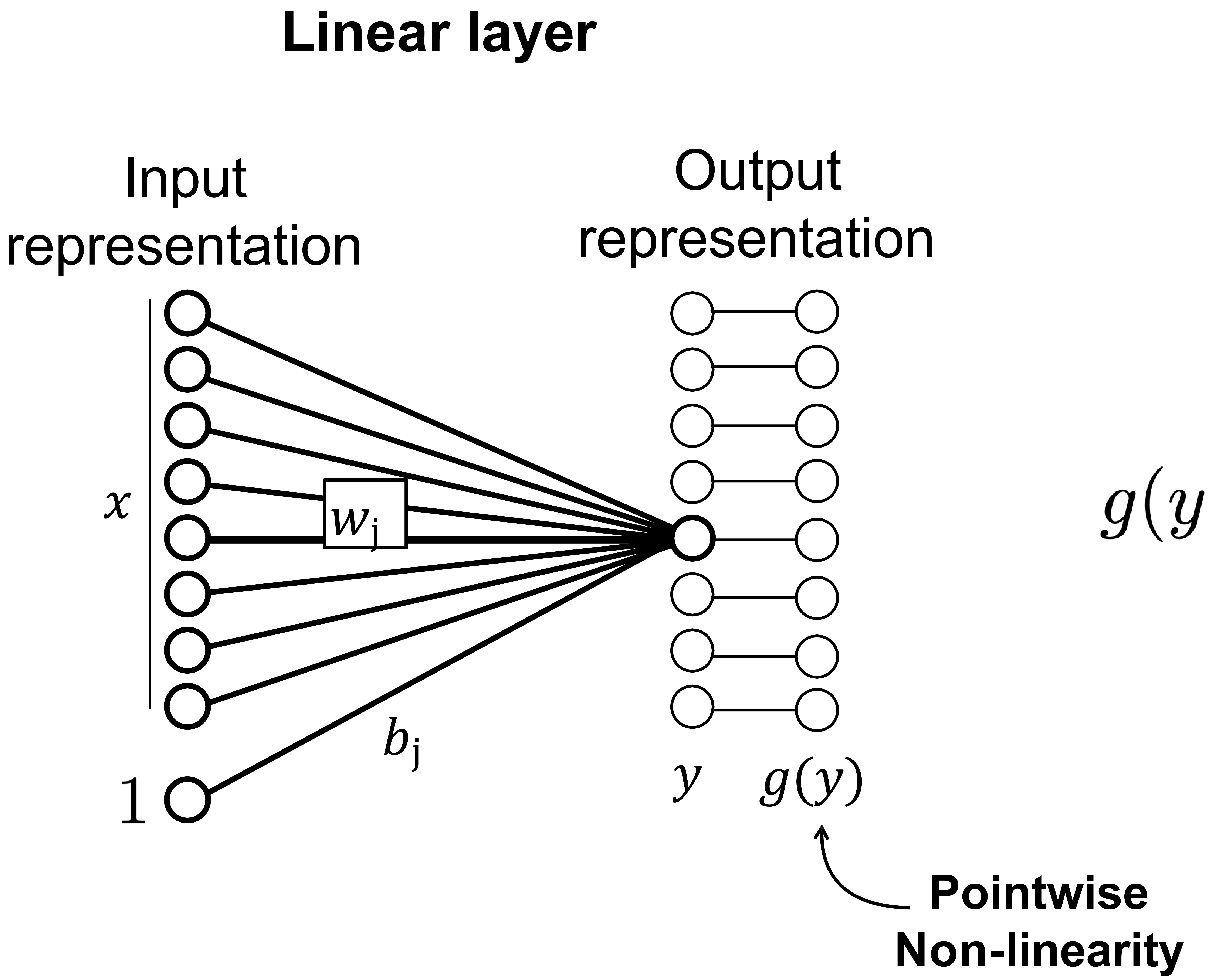


Can be expressed as single linear layer!

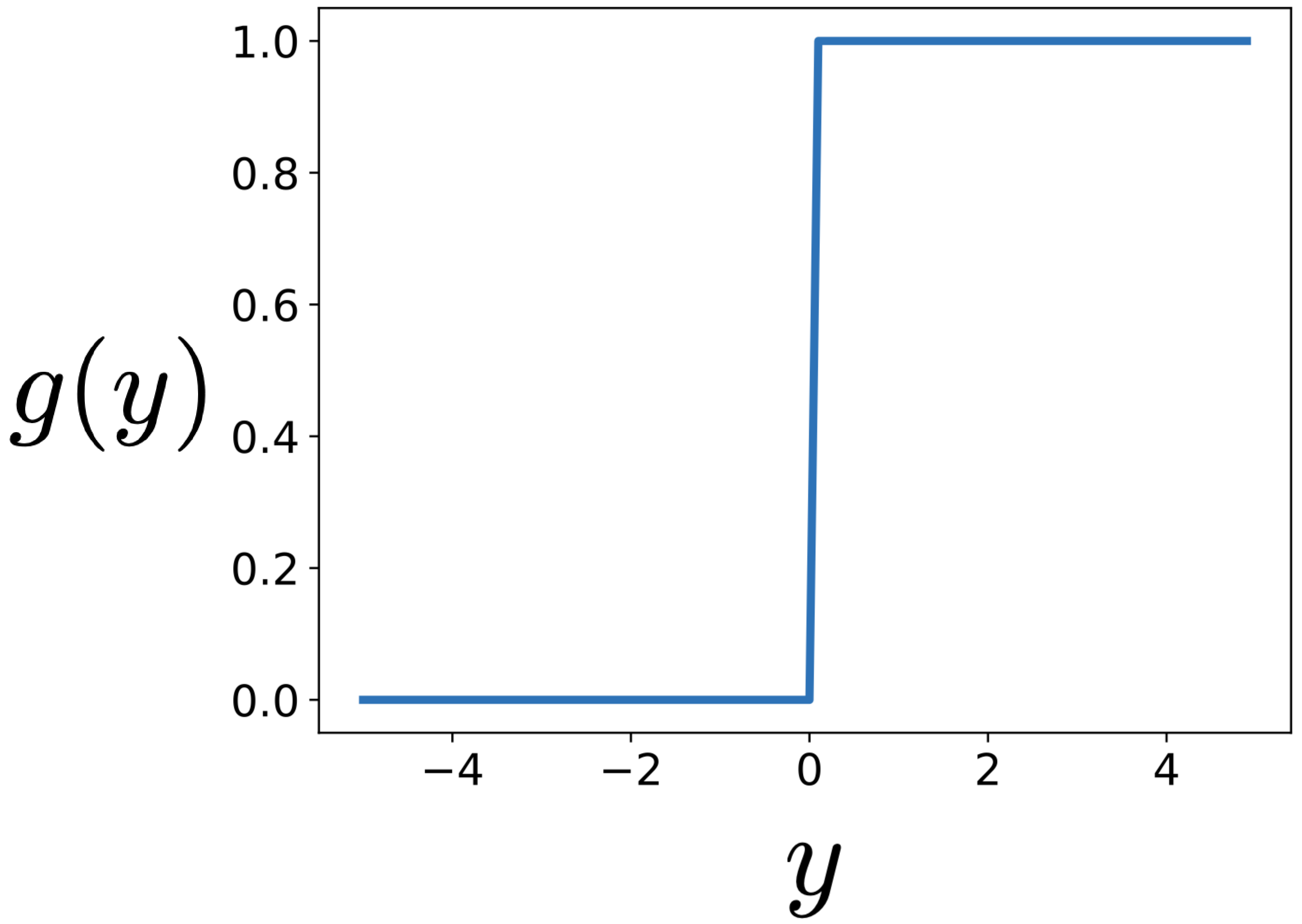
$$\widehat{W}x$$

Limited power: can't solve XOR ☹

Solution: simple nonlinearity



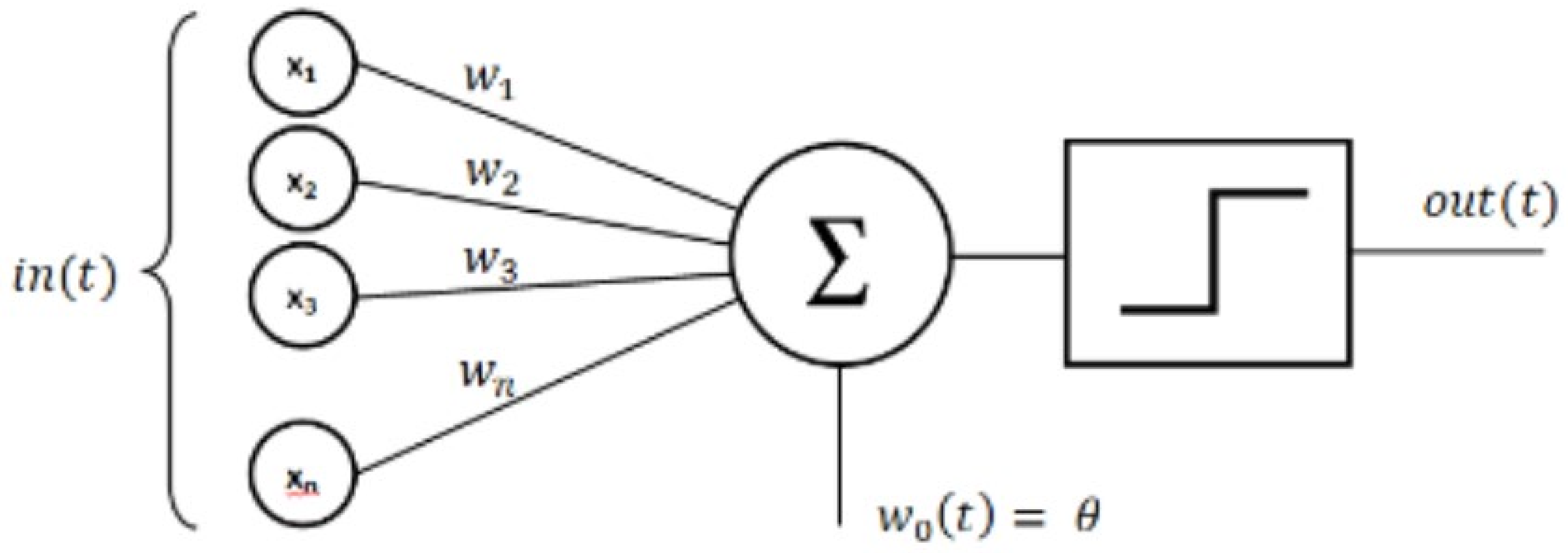
$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$



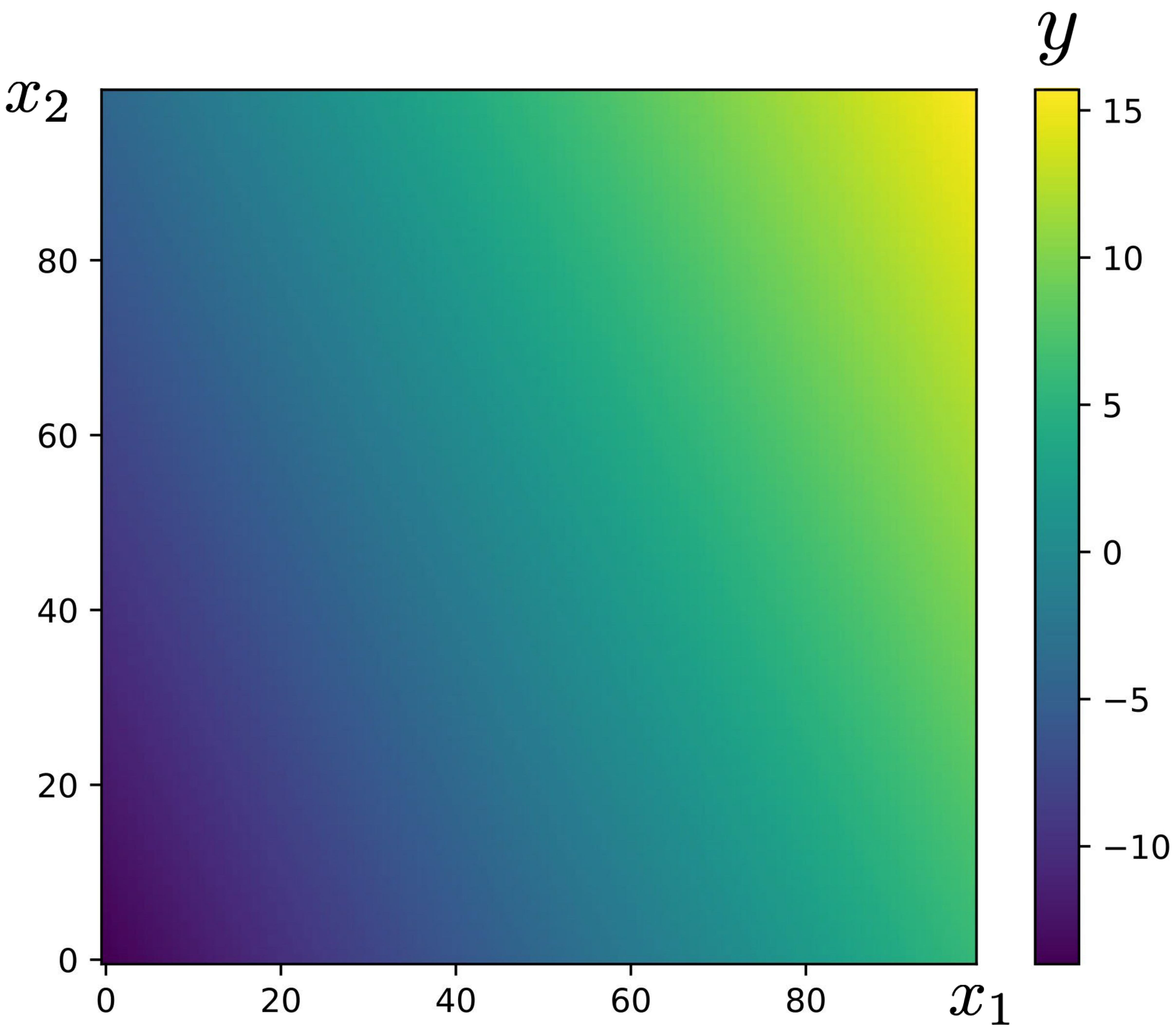
The Perceptron

WHAT PERCEPTRON SOUNDS LIKE

wHat pERceptRoNS ArE



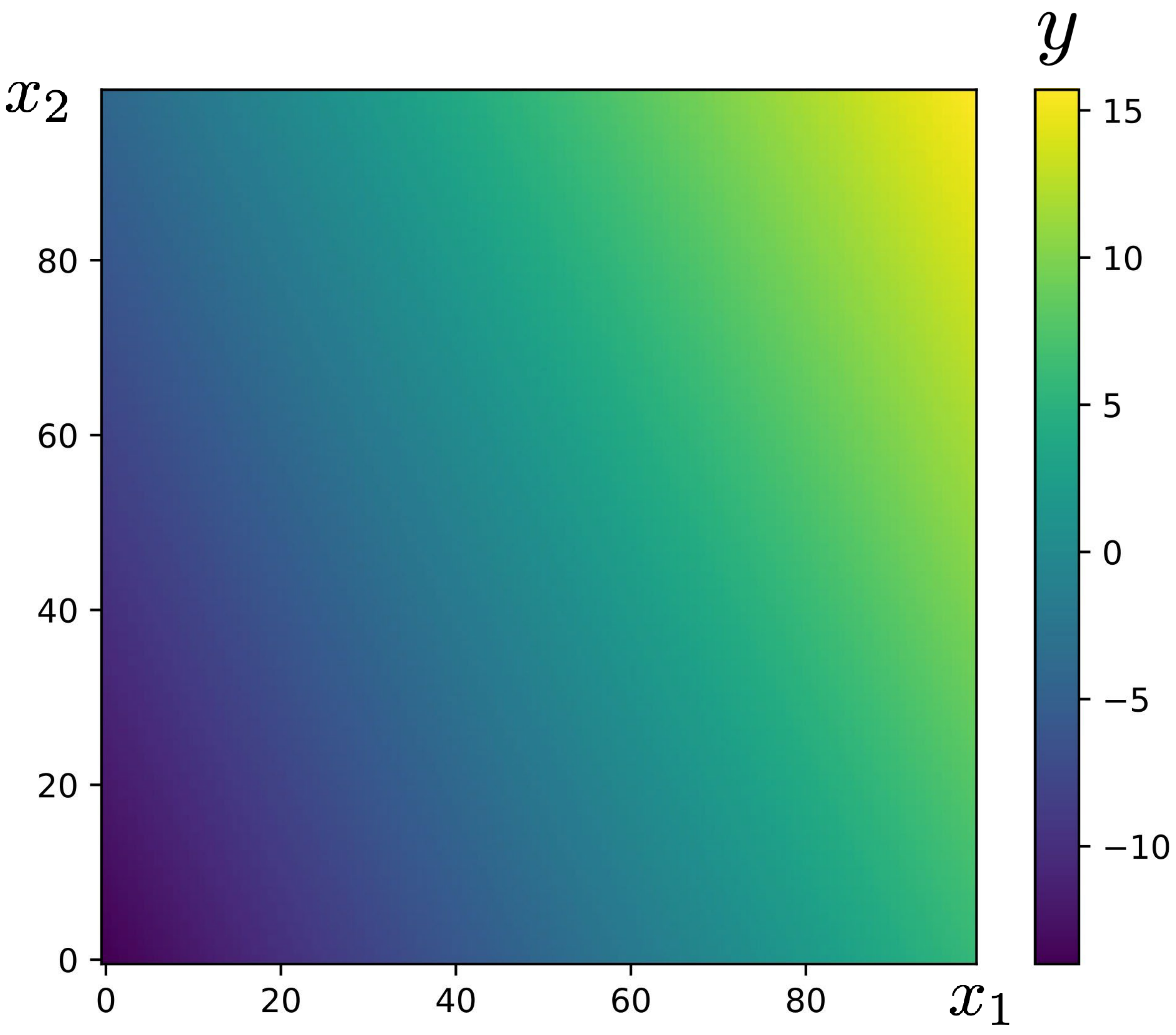
Example: linear classification with a perceptron



$$y = \mathbf{x}^T \mathbf{w} + b$$

Source: Isola, Torralba, Freeman

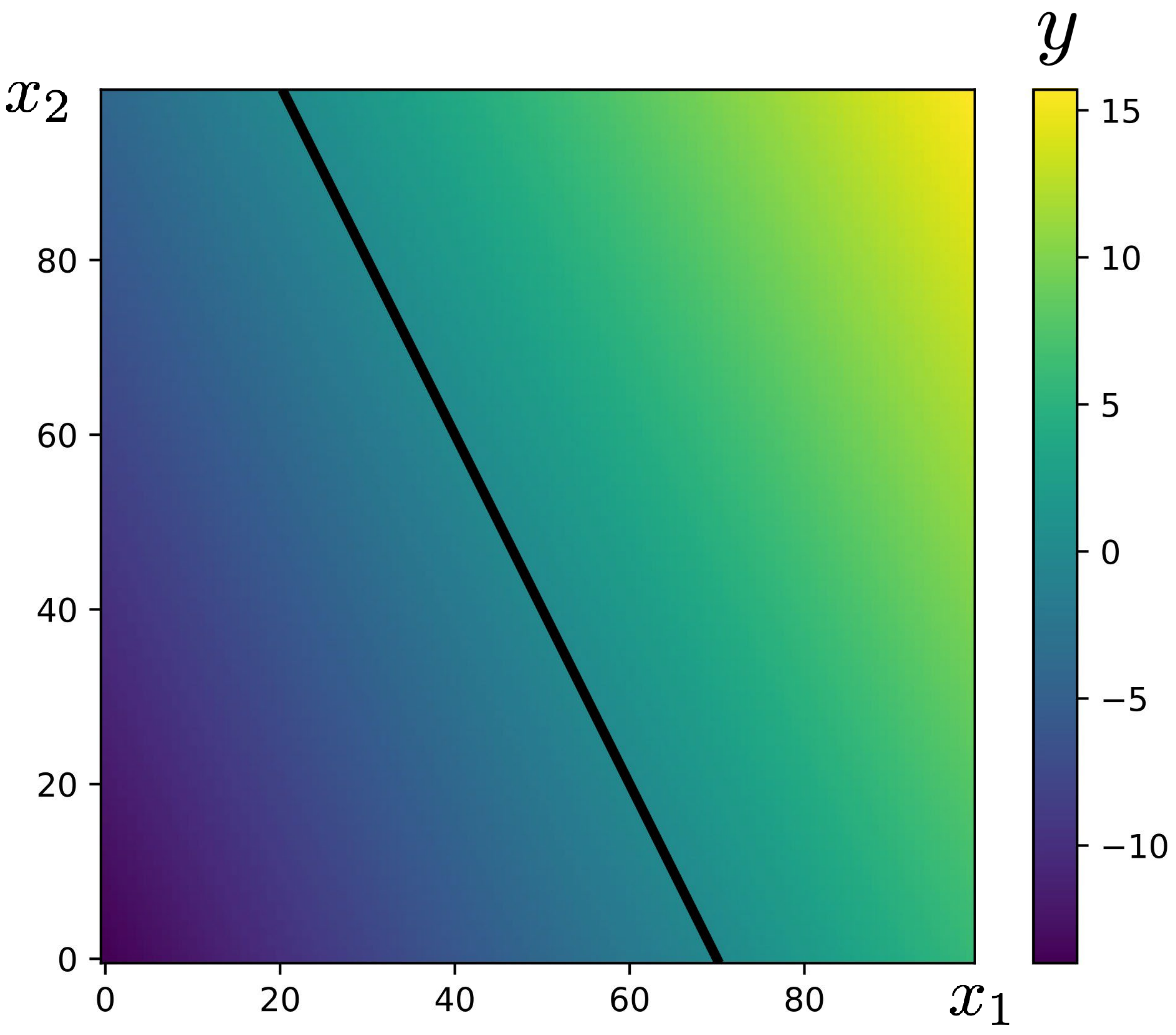
Example: linear classification with a perceptron



$$y = \mathbf{x}^T \mathbf{w} + b$$

$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$

Example: linear classification with a perceptron

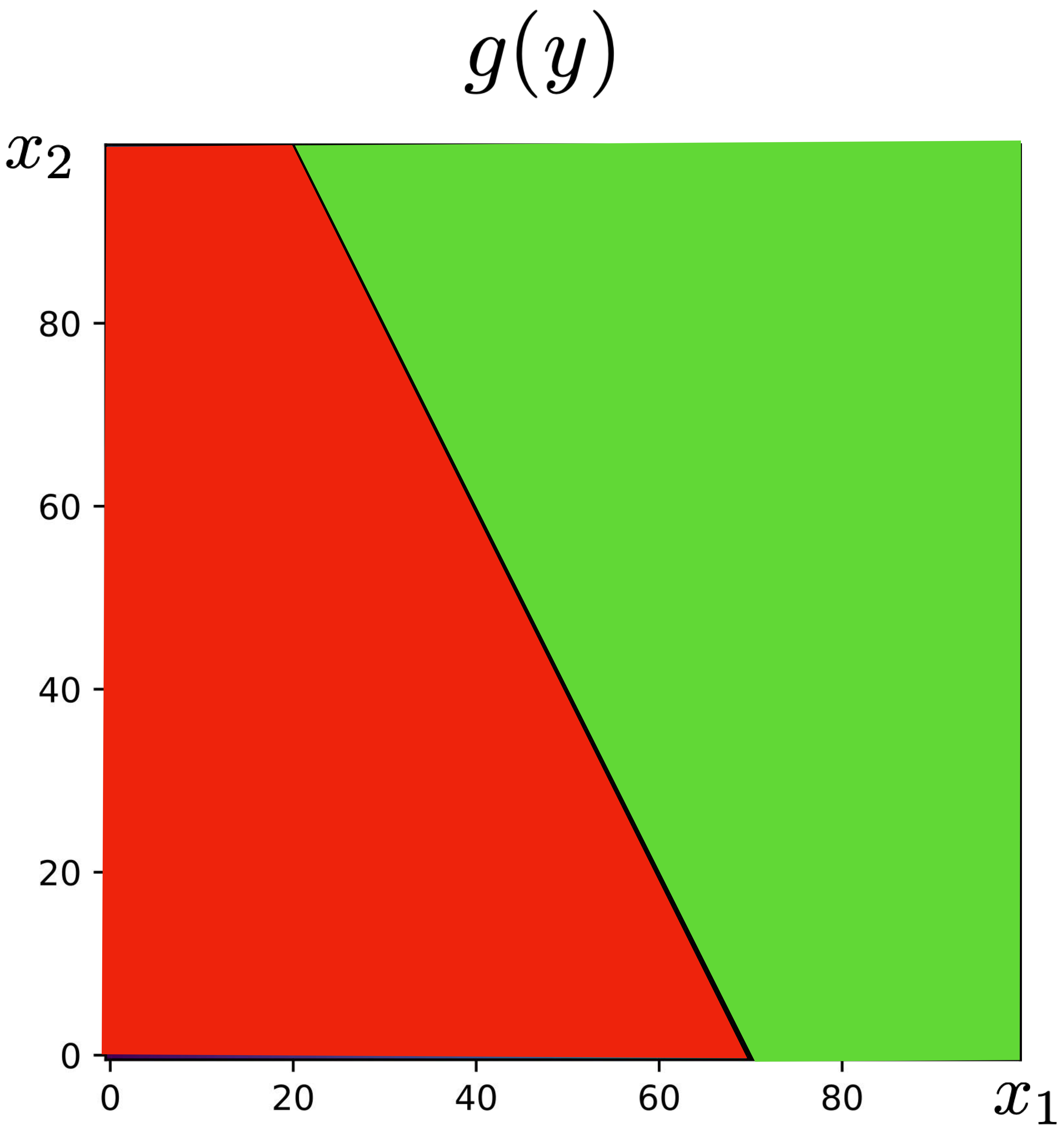


$$y = \mathbf{x}^T \mathbf{w} + b$$

$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$

"when y is greater than 0, set all pixel values to 1 (green), otherwise, set all pixel values to 0 (red)"

Example: linear classification with a perceptron



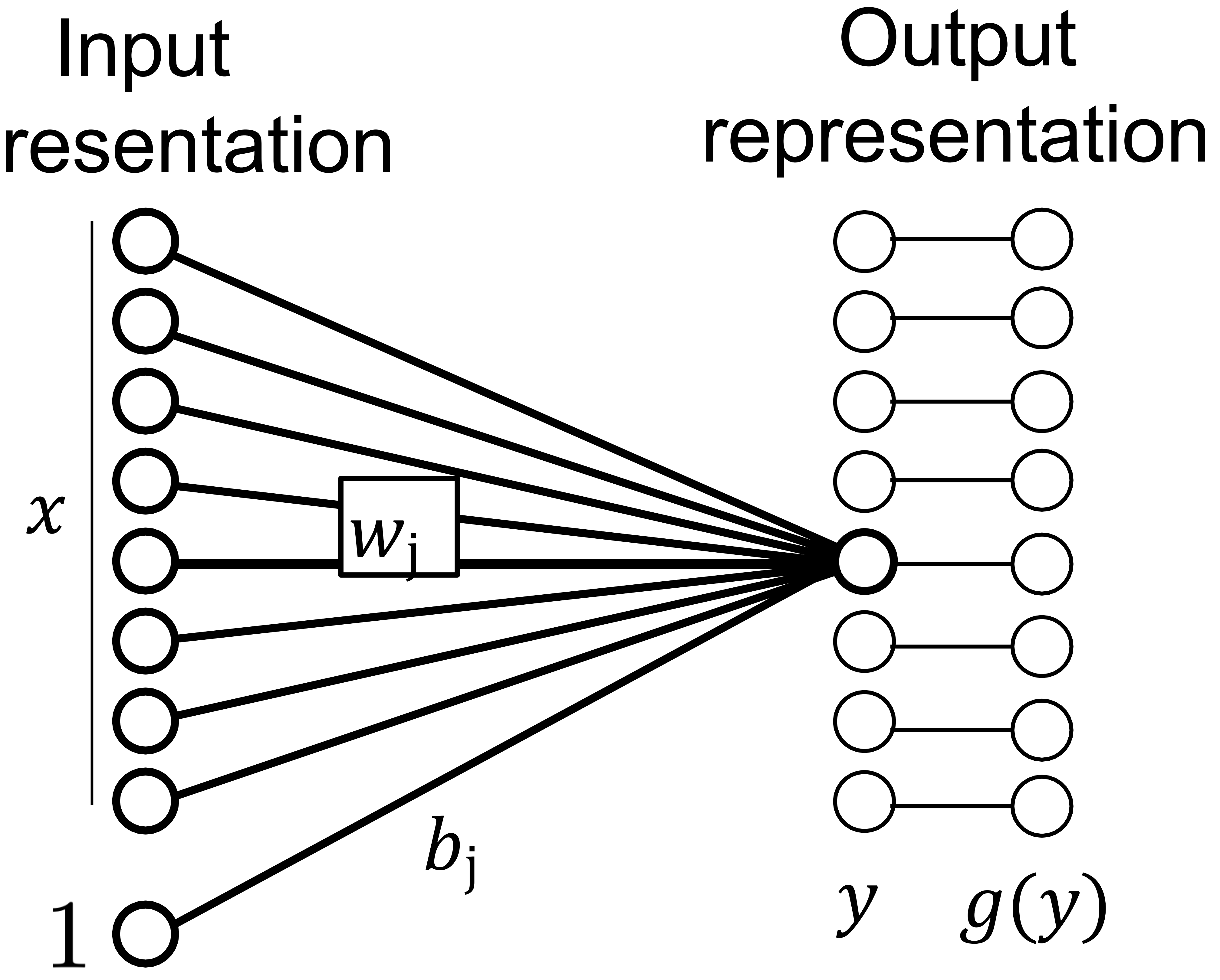
$$y = \mathbf{x}^T \mathbf{w} + b$$

$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$

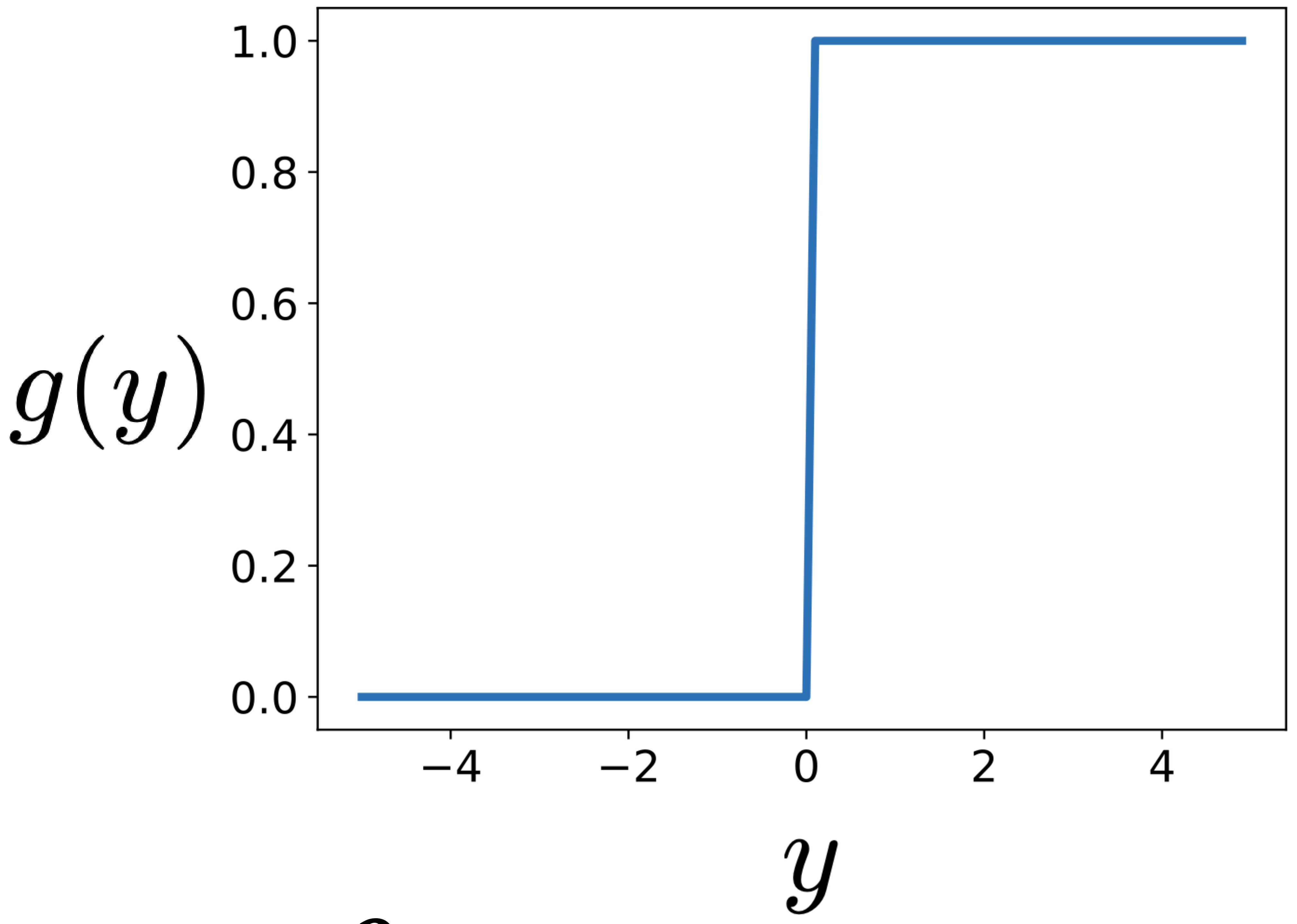
"when y is greater than 0, set all pixel values to 1 (green), otherwise, set all pixel values to 0 (red)"

Computation in a neural net - nonlinearity

Linear layer



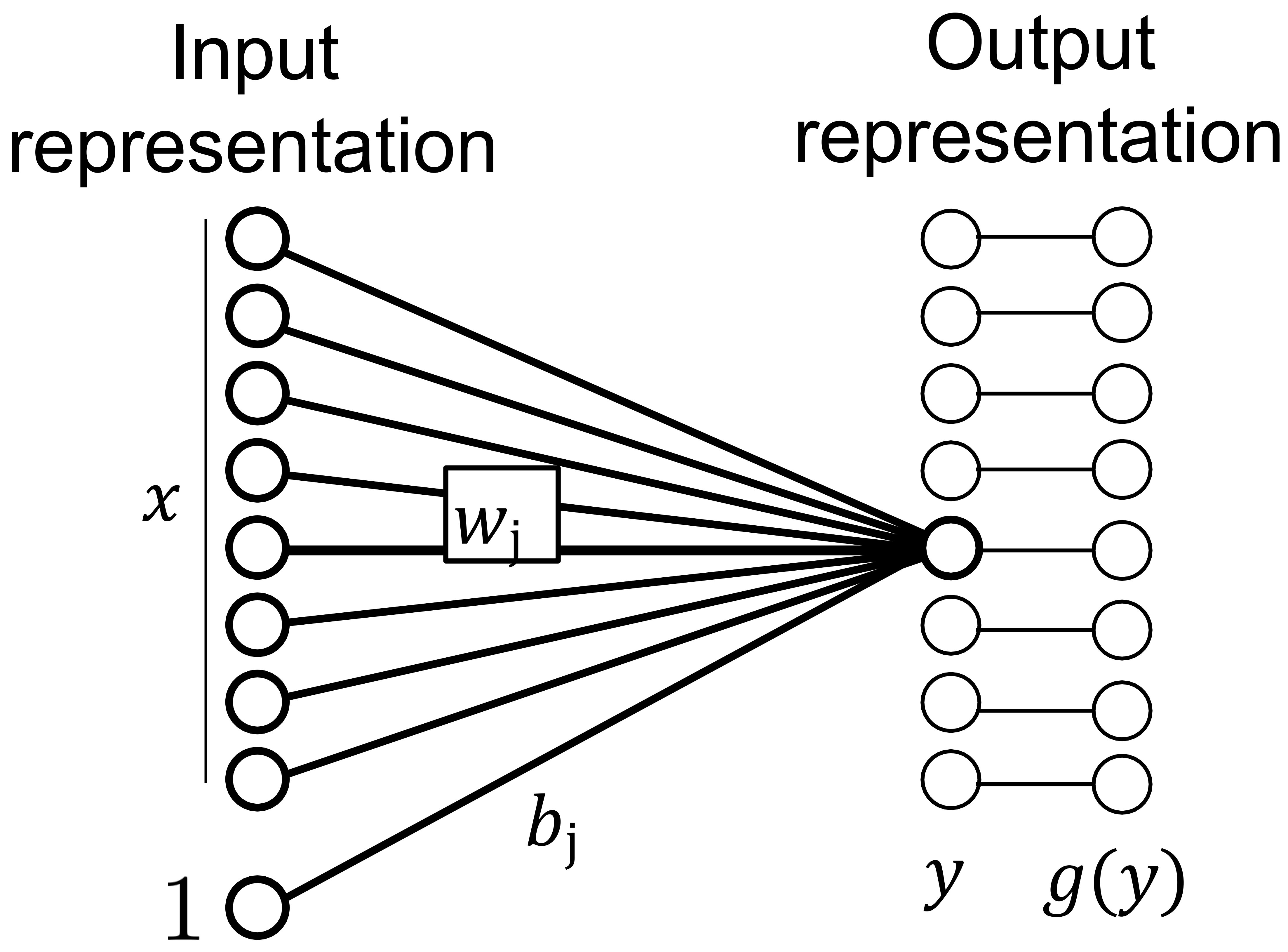
$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$



Can't use with gradient descent, $\frac{\partial}{\partial y} g = 0$

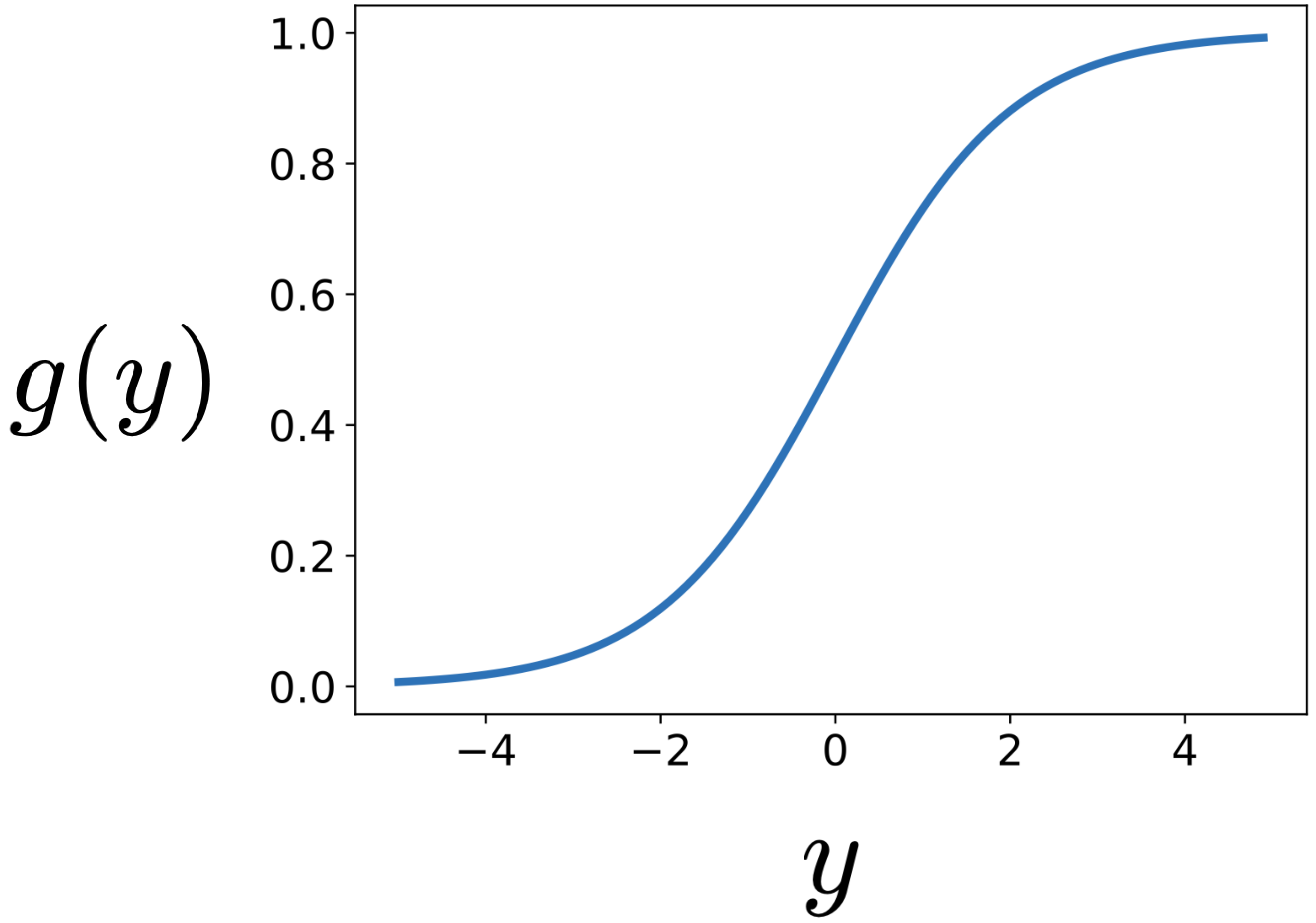
Computation in a neural net - nonlinearity

Linear layer



Sigmoid

$$g(y) = \sigma(y) = \frac{1}{1+e^{-y}}$$

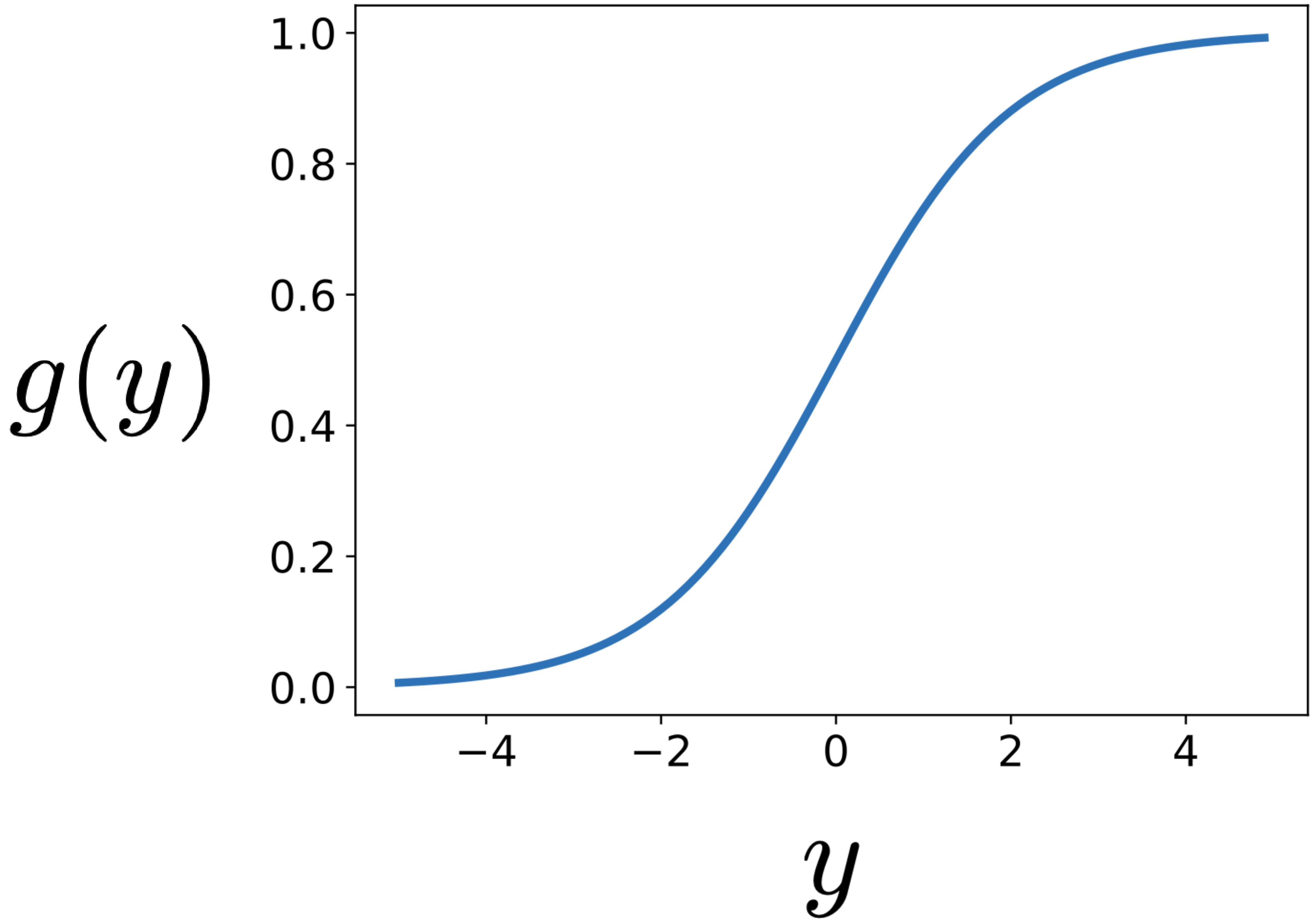


Computation in a neural net - nonlinearity

- Bounded between [0,1]
- Saturation for large +/- inputs
- Gradients go to zero

Sigmoid

$$g(y) = \sigma(y) = \frac{1}{1 + e^{-y}}$$

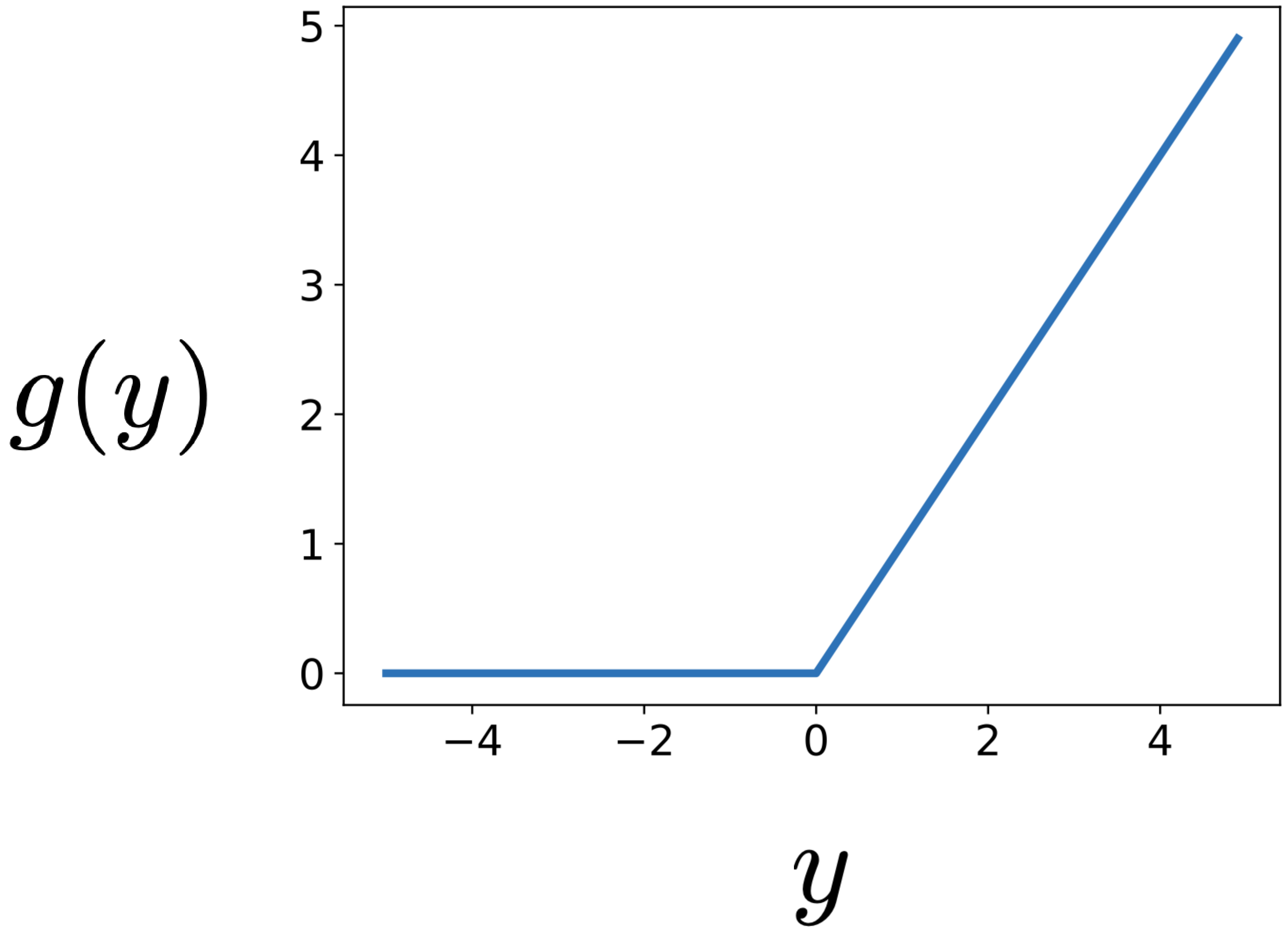


Computation in a neural net — nonlinearity

- Unbounded output (on positive side)
- Efficient to implement: $\frac{\partial g}{\partial y} = \begin{cases} 0, & \text{if} \quad y < 0 \\ 1, & \text{if} \quad y \geq 0 \end{cases}$
- Also seems to help convergence (6x speedup vs. tanh in [Krizhevsky et al. 2012])
- Drawback: if strongly in negative region, unit is dead forever (no gradient).
- Default choice: widely used in current models!

Rectified linear unit (ReLU)

$$g(y) = \max(0, y)$$



Computation in a neural net — nonlinearity

- where α is small (e.g., 0.02)

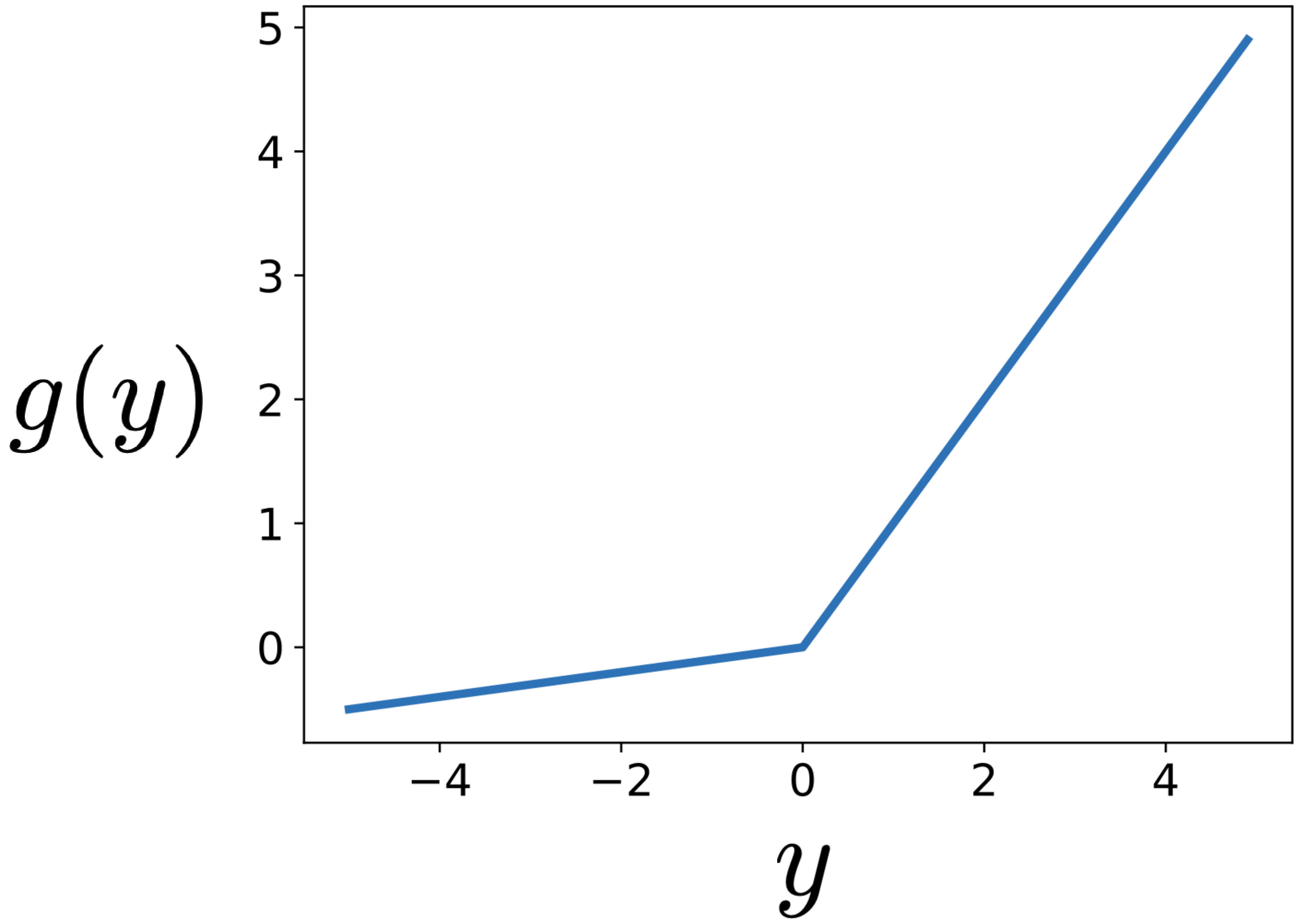
- Efficient to implement:

- Has non-zero gradients everywhere (unlike ReLU)

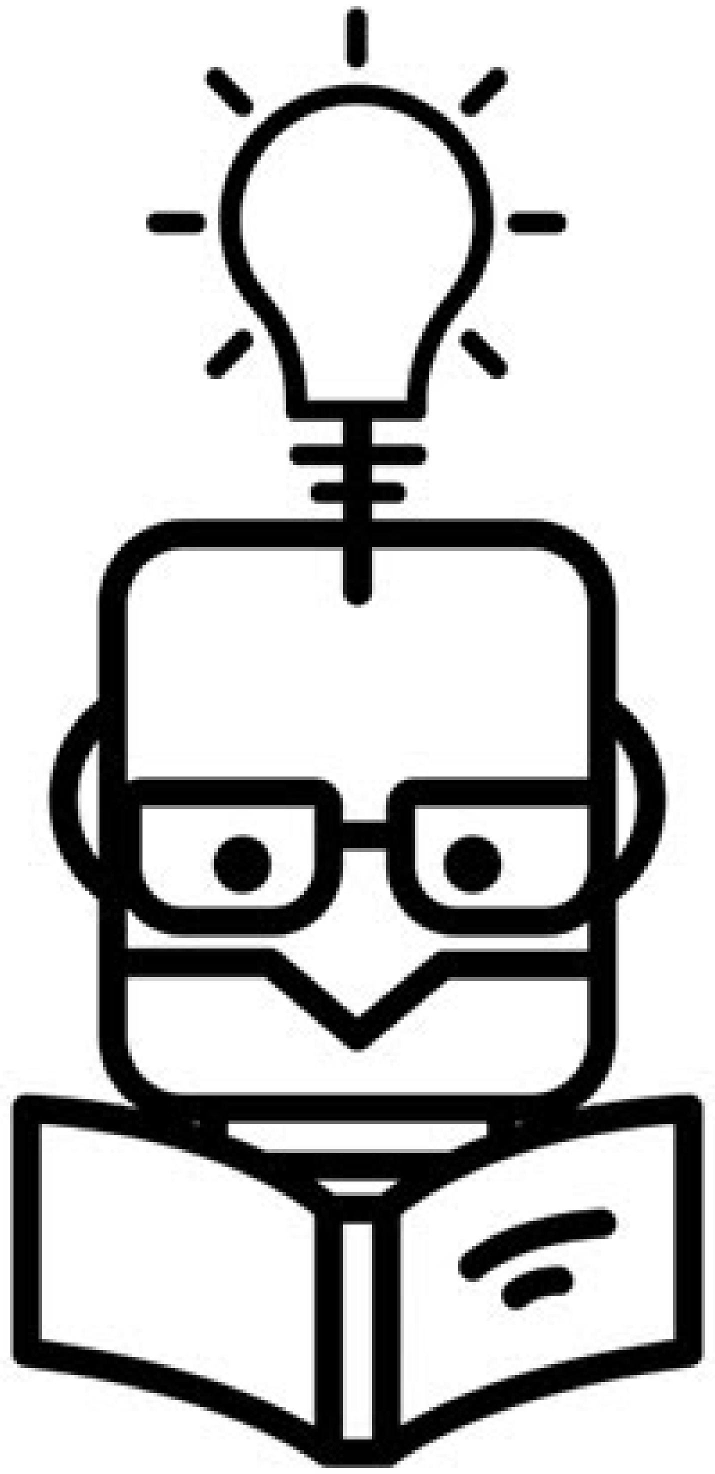
$$\frac{\partial g}{\partial y} = \begin{cases} -a, & \text{if} \quad y < 0 \\ 1, & \text{if} \quad y \geq 0 \end{cases}$$

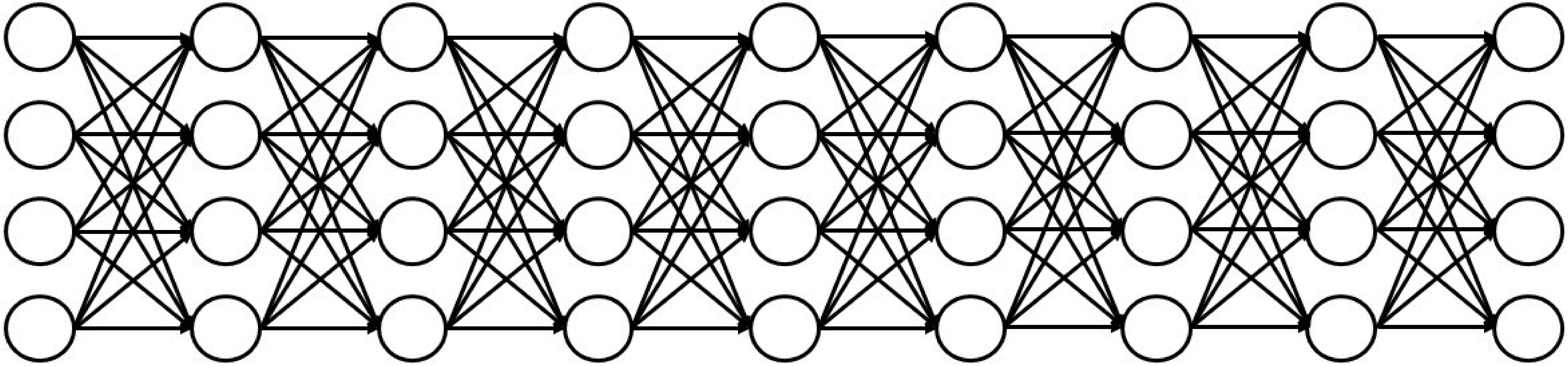
Leaky ReLU

$$g(y) = \begin{cases} \max(0, y), & \text{if} \quad y \geq 0 \\ a \min(0, y), & \text{if} \quad y < 0 \end{cases}$$

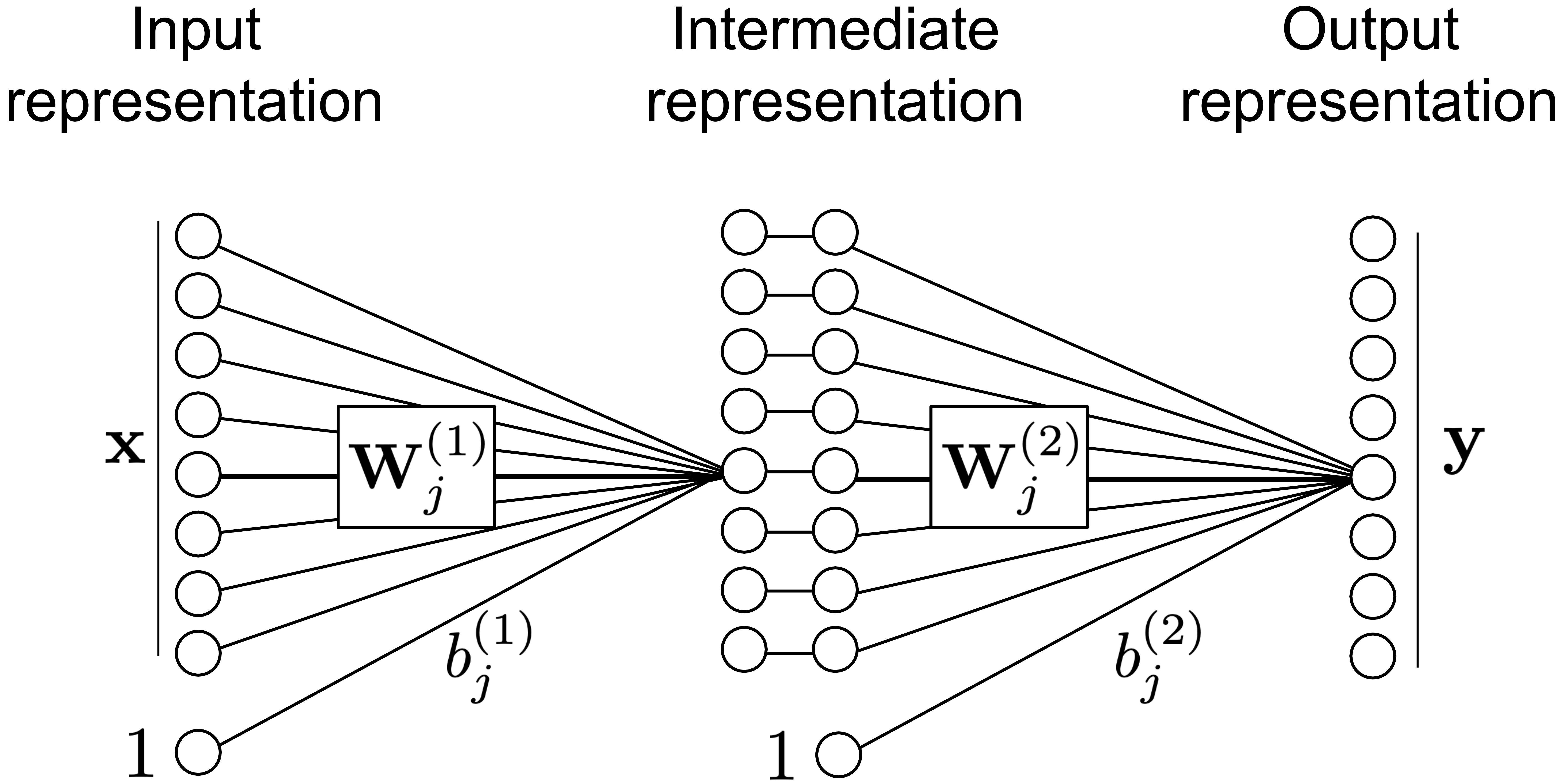


DEEP Neural Nets?



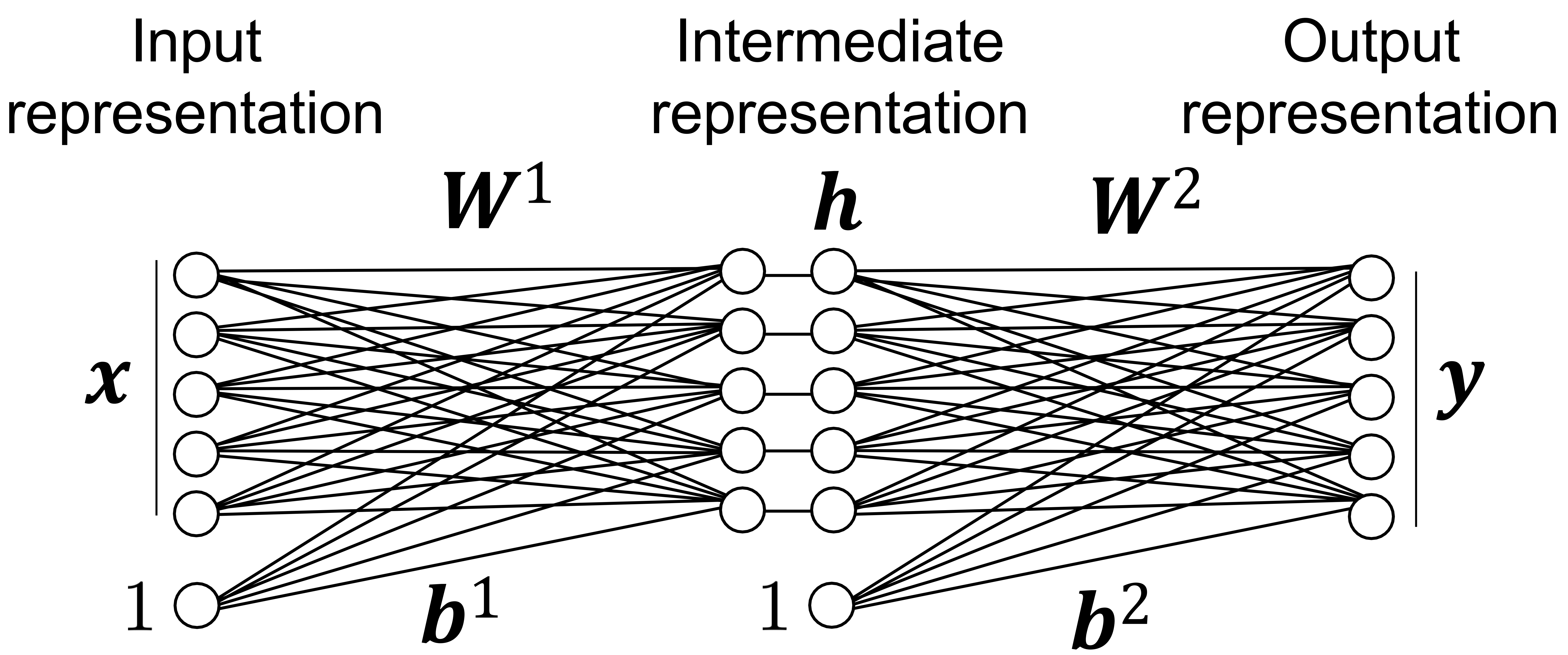


Stacking layers



$\mathbf{h}$ = “hidden units”

Stacking layers



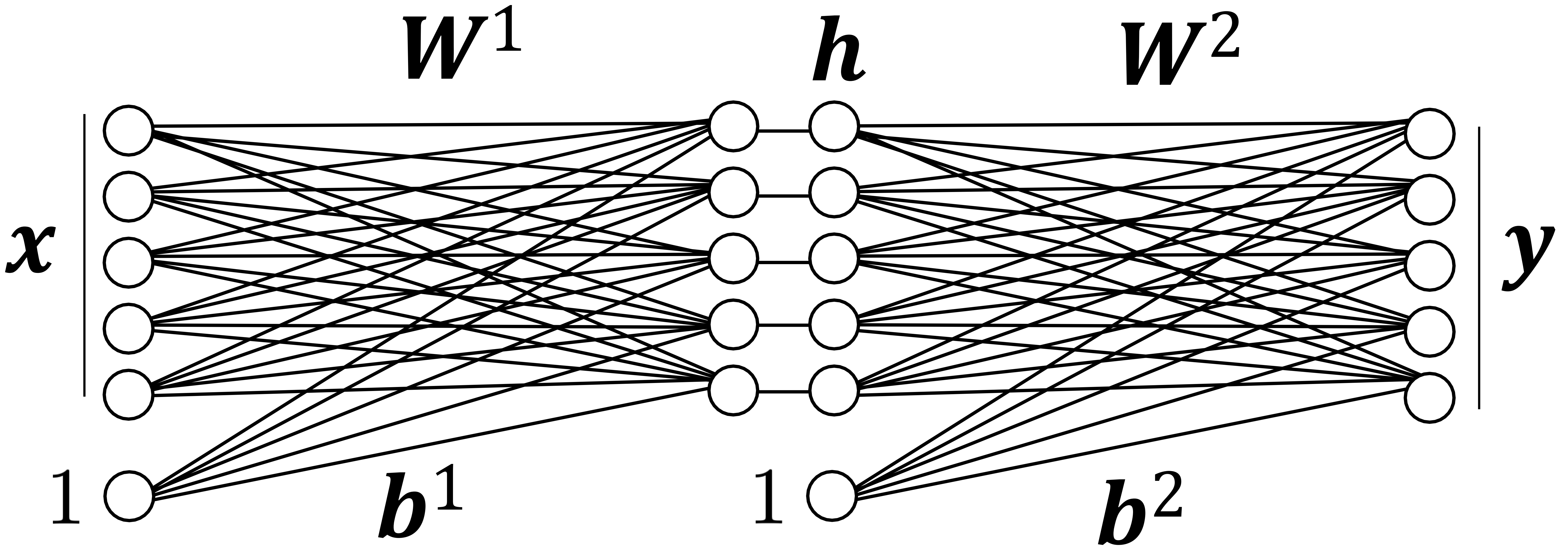
$$\boldsymbol{h} = g(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1) \qquad \boldsymbol{y} = g(\boldsymbol{W}^2\boldsymbol{h} + \boldsymbol{b}^2)$$

ReLU

$$\theta = \{\boldsymbol{W}^1, \dots, \boldsymbol{W}^L, \boldsymbol{b}^1, \dots, \boldsymbol{b}^L\}$$

Stacking layers

Input representation | Intermediate representation | Output representation



positive

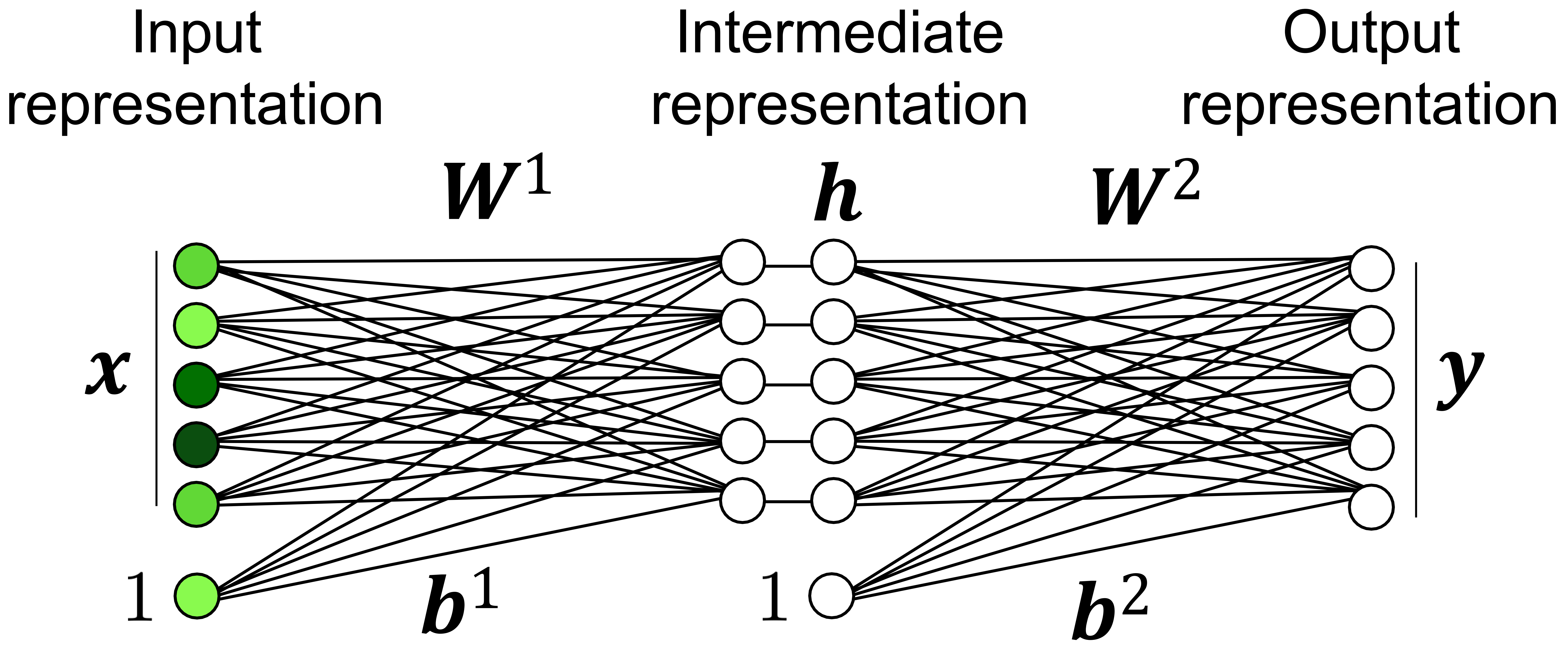
negative

$$\boldsymbol{h} = g(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1) \quad \boldsymbol{y} = g(\boldsymbol{W}^2\boldsymbol{h} + \boldsymbol{b}^2)$$

ReLU

$$\theta = \{\boldsymbol{W}^1, \dots, \boldsymbol{W}^L, \boldsymbol{b}^1, \dots, \boldsymbol{b}^L\}$$

Stacking layers



positive

negative

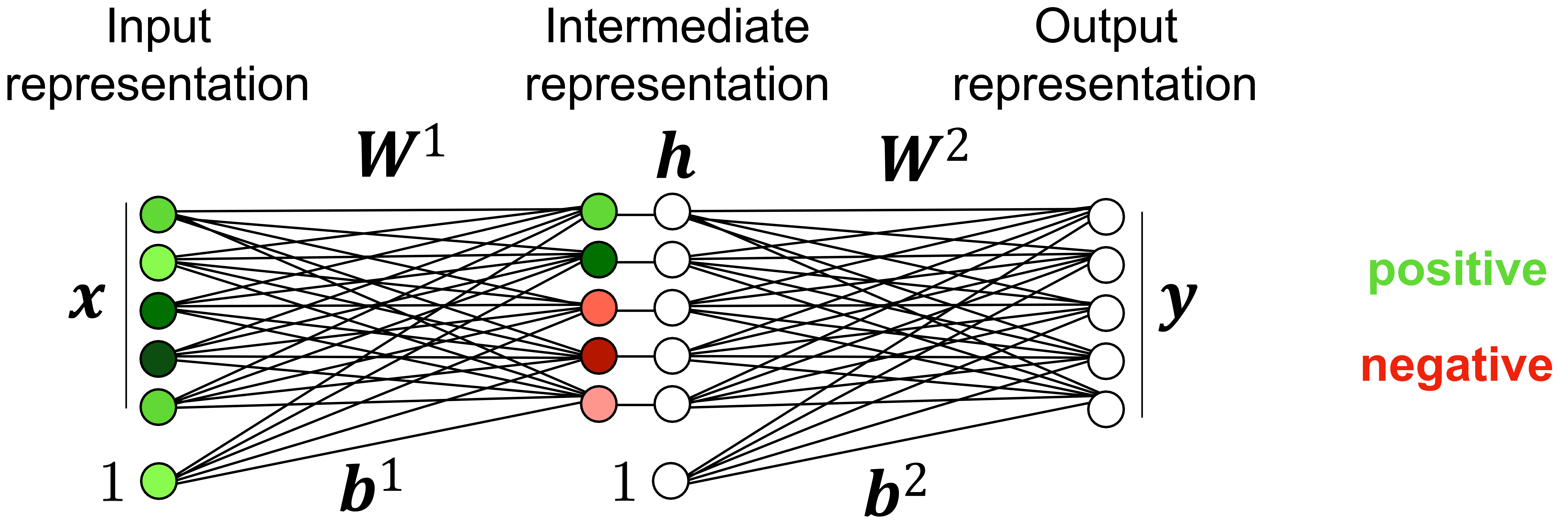
$$\boldsymbol{h} = g(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1) \quad \boldsymbol{y} = g(\boldsymbol{W}^2\boldsymbol{h} + \boldsymbol{b}^2)$$

ReLU

$$\theta = \{\boldsymbol{W}^1, \ldots, \boldsymbol{W}^L, \boldsymbol{b}^1, \ldots, \boldsymbol{b}^L\}$$

Source: Isola, Torralba, Freeman

Stacking layers

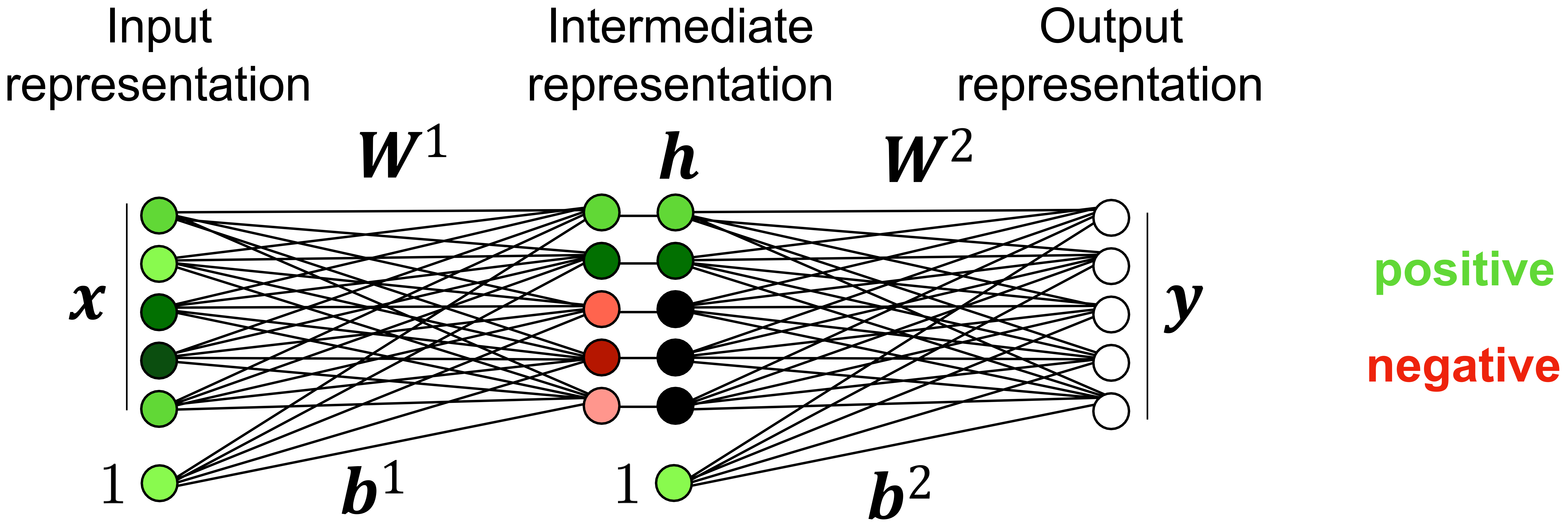


$$\boldsymbol{h} = g(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1) \quad \boldsymbol{y} = g(\boldsymbol{W}^2\boldsymbol{h} + \boldsymbol{b}^2)$$

ReLU

$$\theta = \{\boldsymbol{W}^1, \ldots, \boldsymbol{W}^L, \boldsymbol{b}^1, \ldots, \boldsymbol{b}^L\}$$

Stacking layers

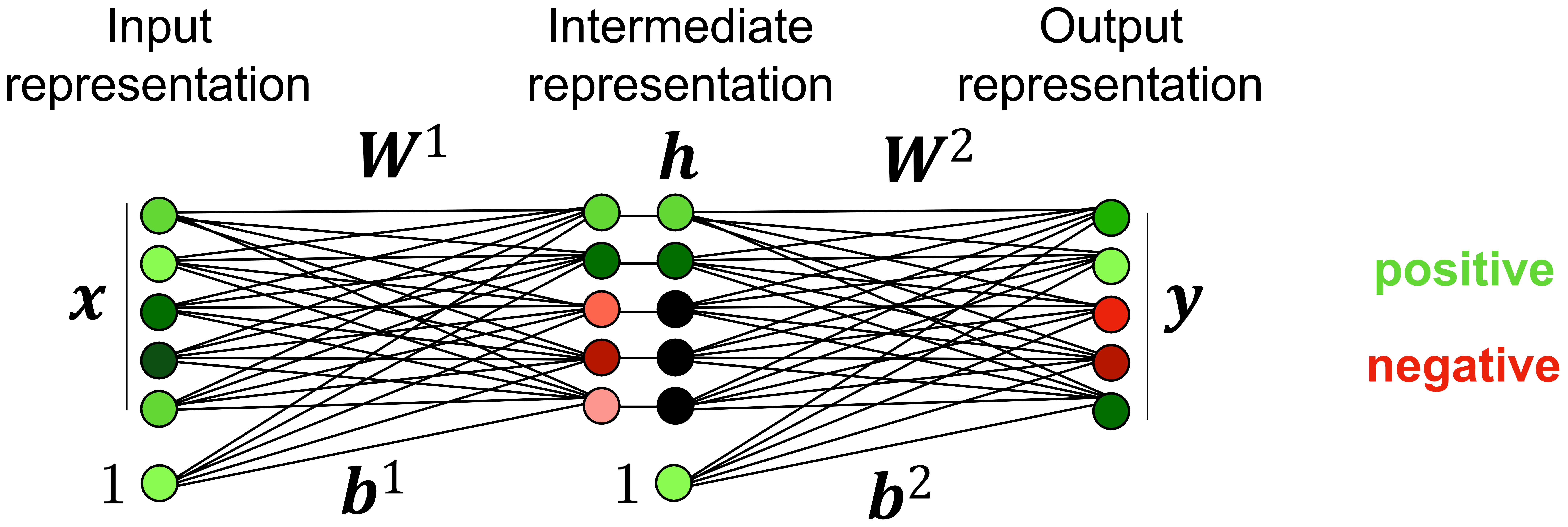


$$\boldsymbol{h} = g(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1) \qquad \boldsymbol{y} = g(\boldsymbol{W}^2\boldsymbol{h} + \boldsymbol{b}^2)$$

ReLU

$$\theta = \{\boldsymbol{W}^1, \dots, \boldsymbol{W}^L, \boldsymbol{b}^1, \dots, \boldsymbol{b}^L\}$$

Stacking layers

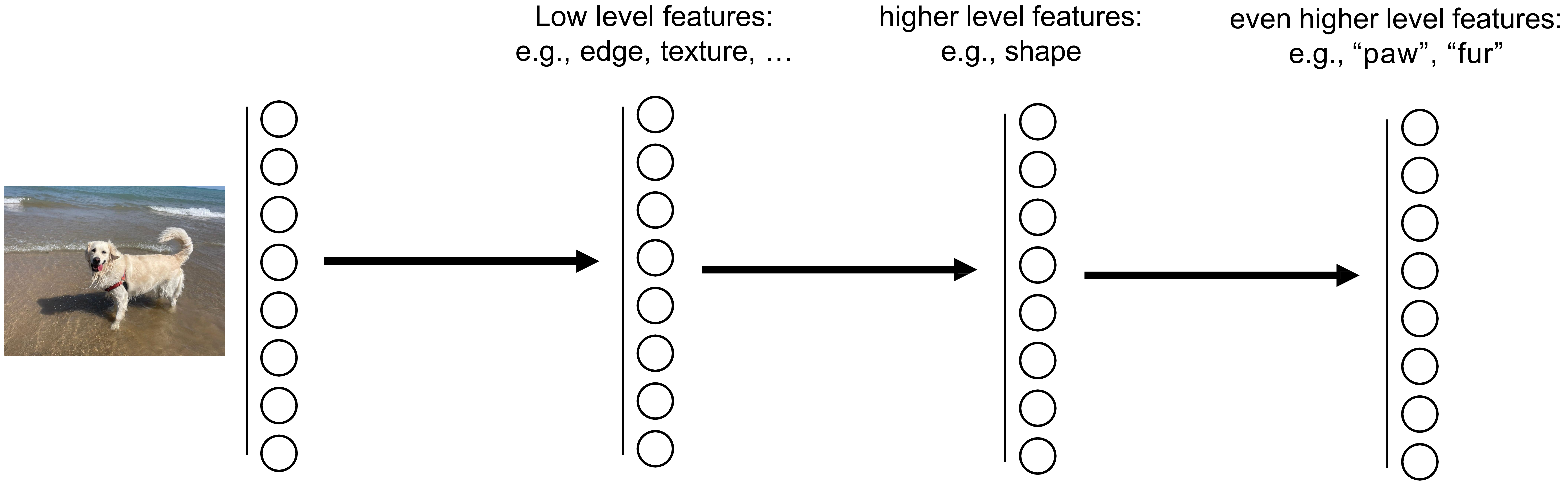


$$\boldsymbol{h} = g(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1) \quad \boldsymbol{y} = g(\boldsymbol{W}^2\boldsymbol{h} + \boldsymbol{b}^2)$$

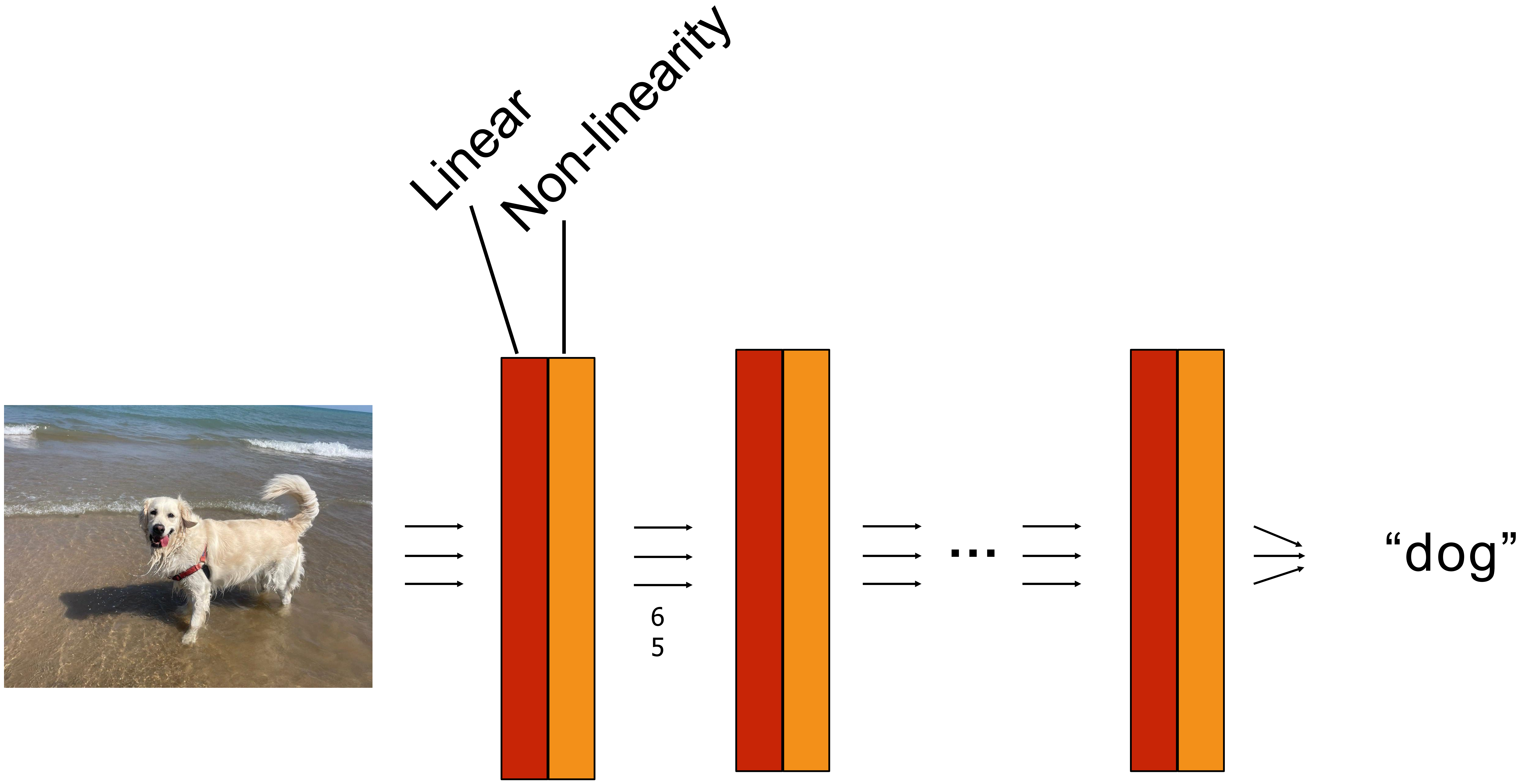
ReLU

$$\theta = \{\boldsymbol{W}^1, \ldots, \boldsymbol{W}^L, \boldsymbol{b}^1, \ldots, \boldsymbol{b}^L\}$$

Stacking layers - What's actually happening?



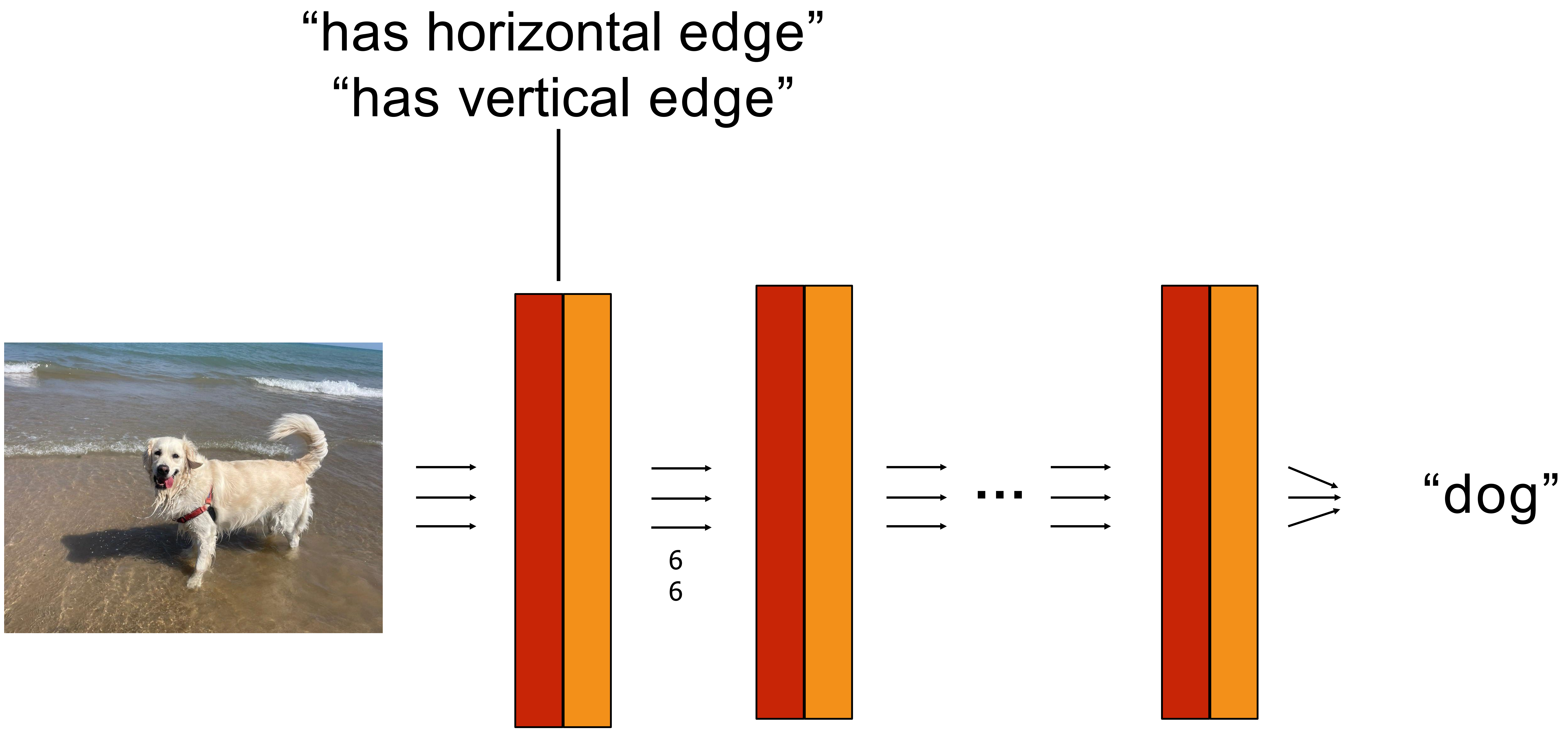
Deep nets



$$f(x) = f_L(\ldots f_3(f_2(f_1(x))))$$

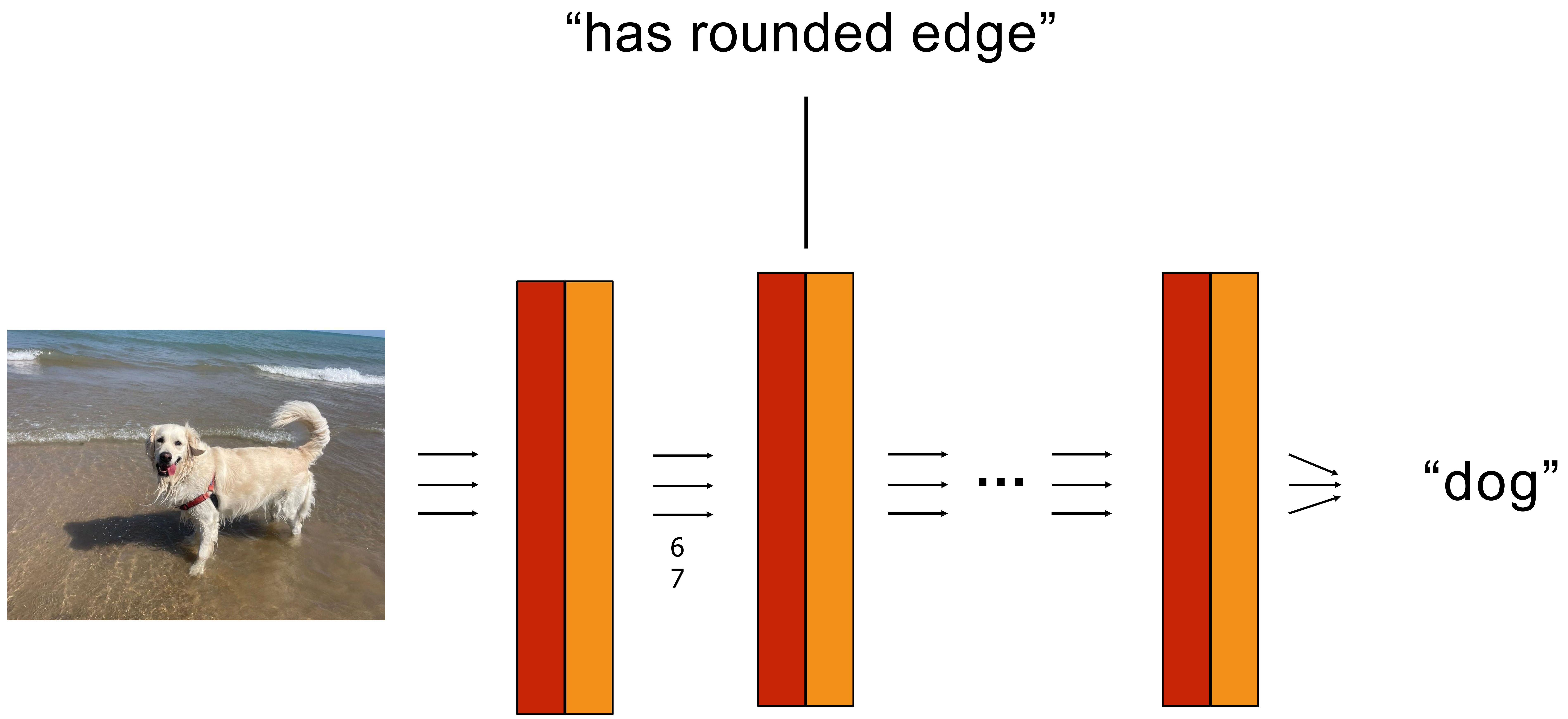
Source: Isola, Torralba, Freeman

Deep nets - Intuition



Source: Isola, Torralba, Freeman

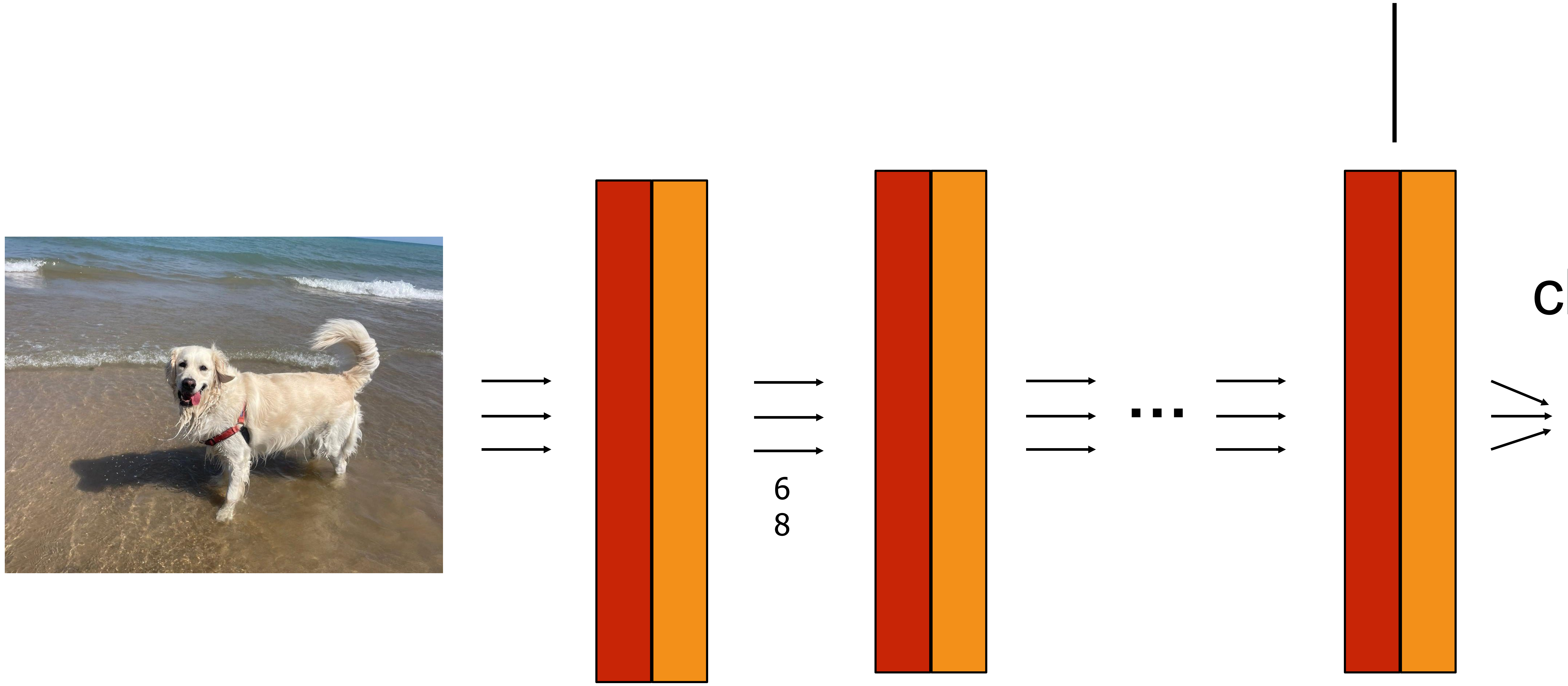
Deep nets - Intuition



Source: Isola, Torralba, Freeman

Deep nets - Intuition

“has white fur”
“has paw”
etc



How do we make a classification?

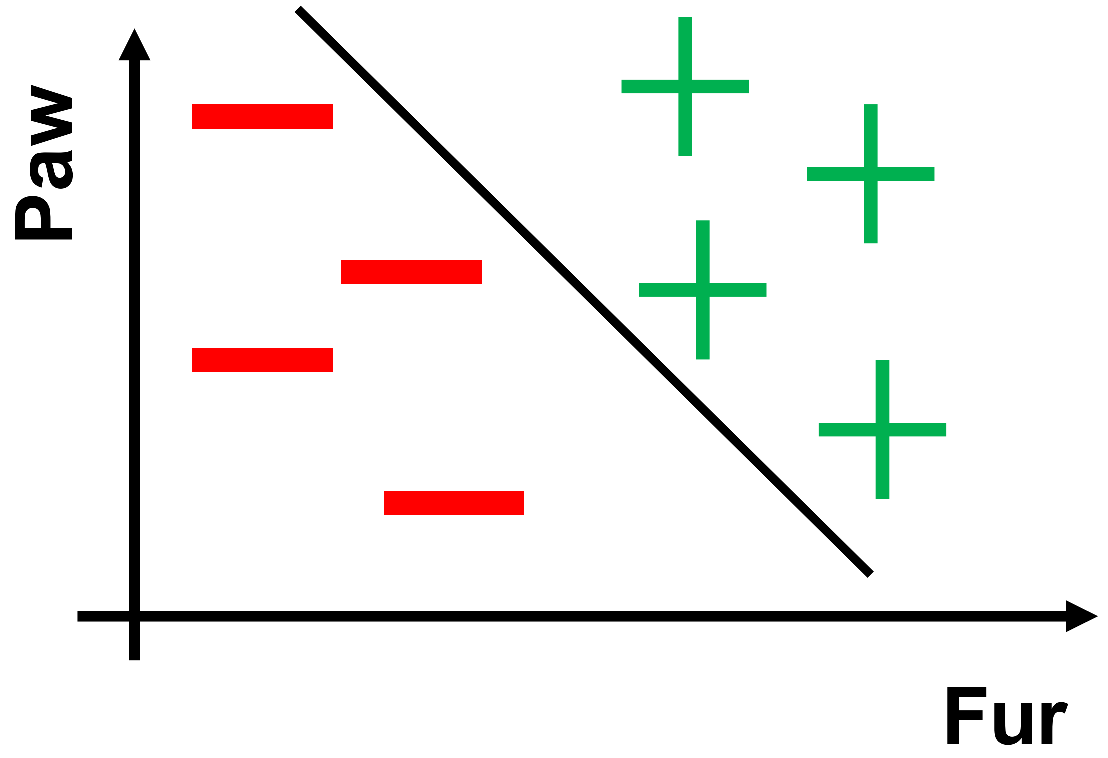
“dog”

Source: Isola, Torralba, Freeman

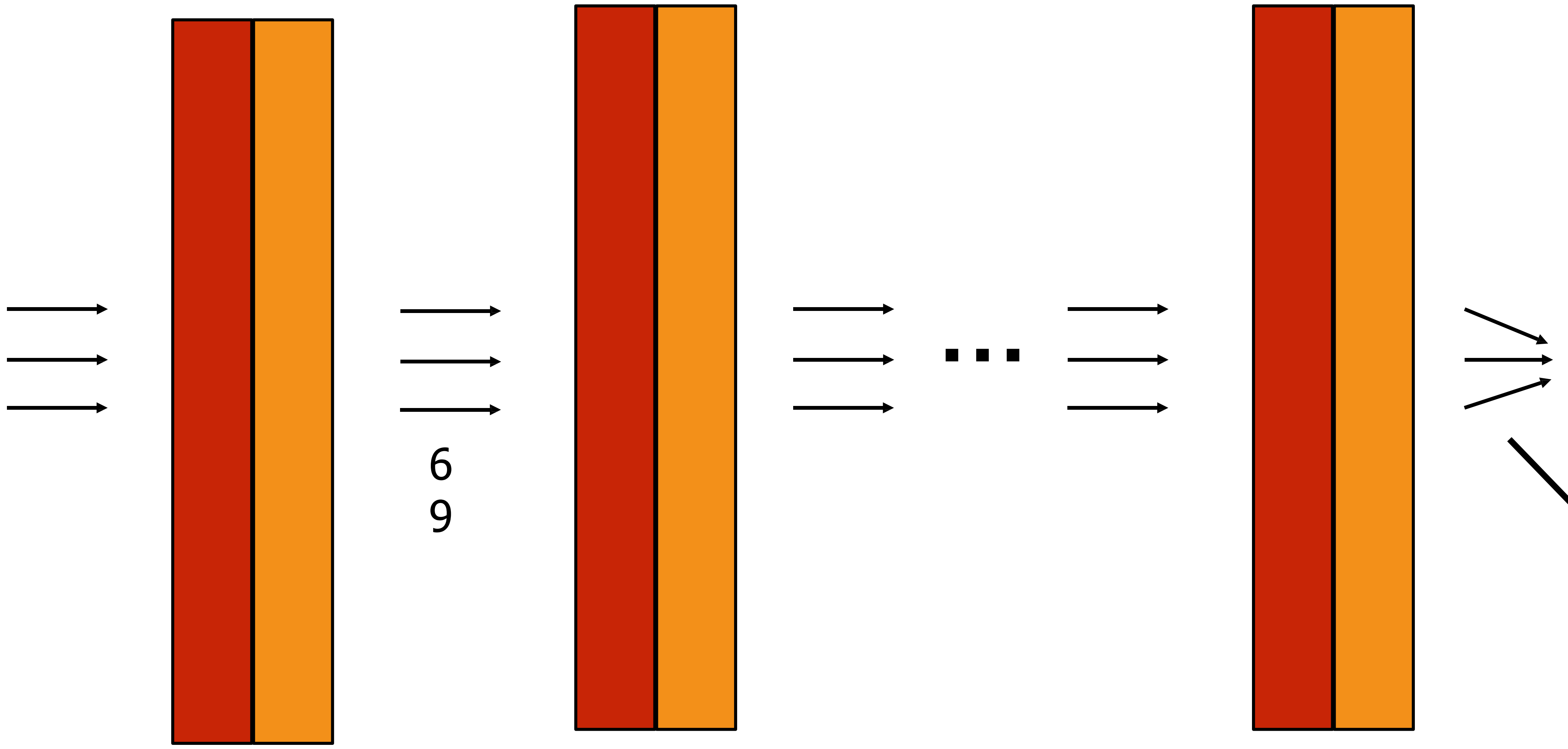
Deep nets - Intuition

Recall:

Feature Space



"has white fur"
"has paw"
etc



"dog"

Classify

Source: Isola, Torralba, Freeman

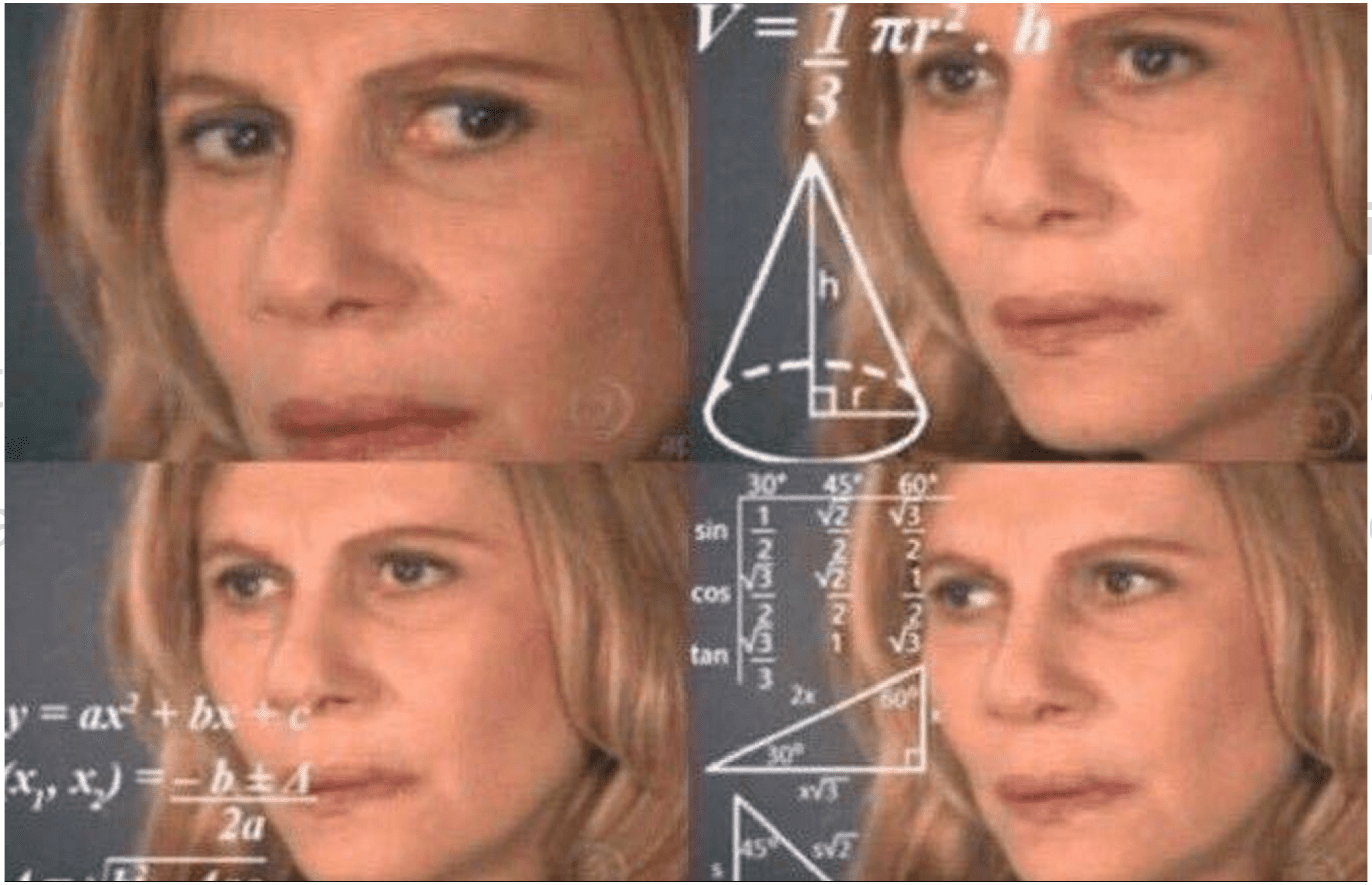
Computation has a simple form

- Composition of linear functions with nonlinearities in between
- E.g. matrix multiplications with ReLU, $max(0, \mathbf{x})$ afterwards
- Do a matrix multiplication, set all negative values to 0, repeat

But where do we get the weights from?

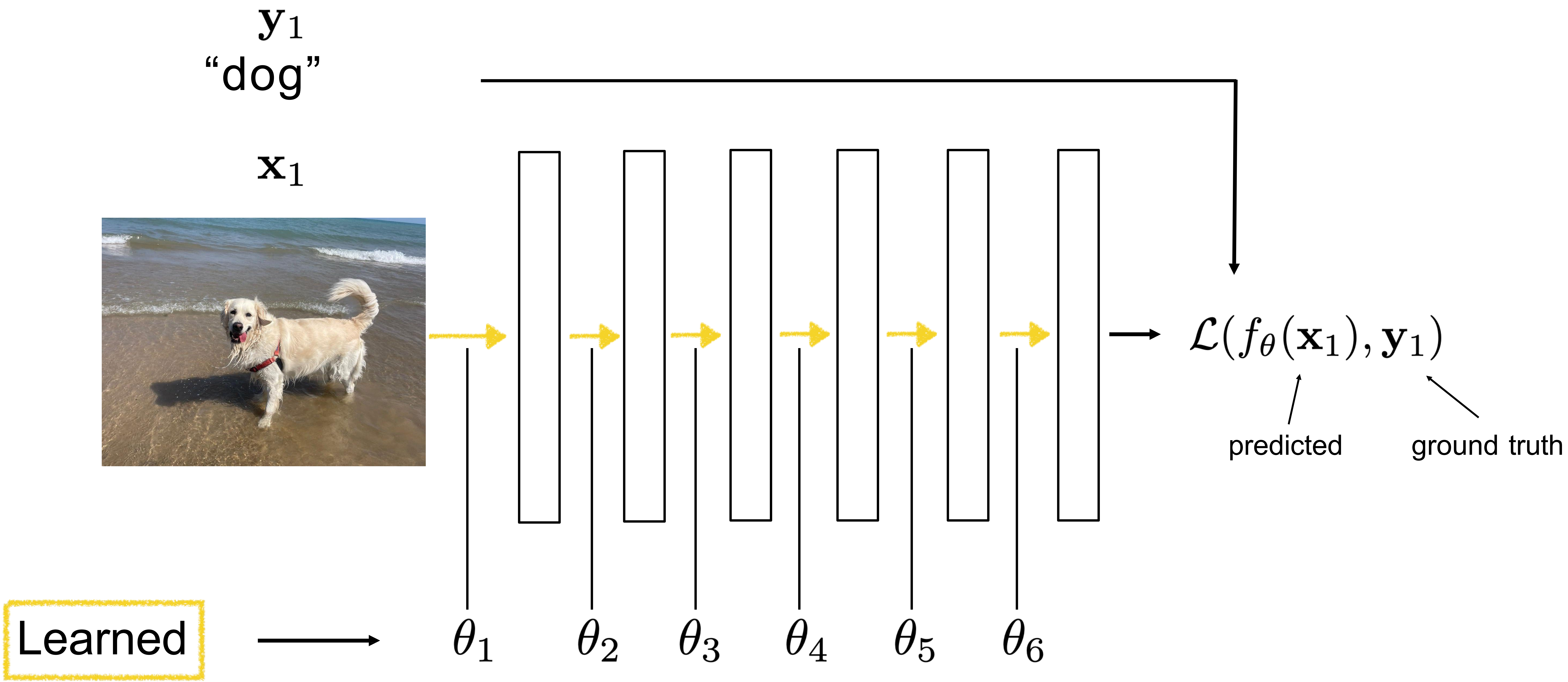
Computation has a simple form

- Compos… in between
- E.g. mat… afterwards
- Do a ma… o 0, repeat



But where do we get the weights from?

How would we learn the parameters?

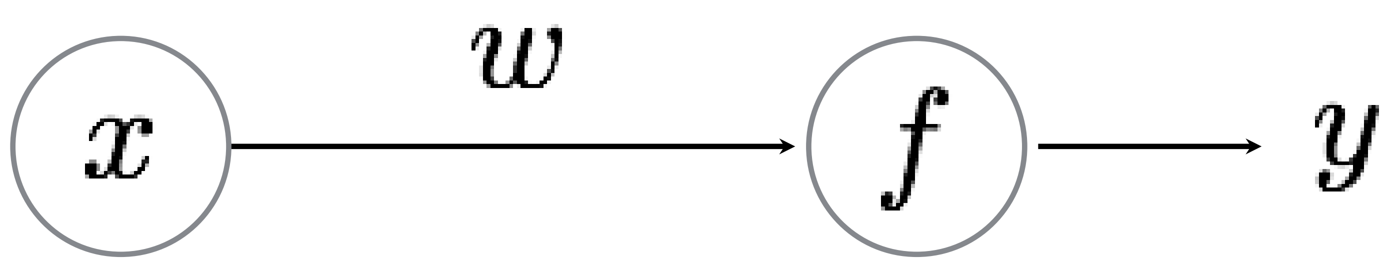


$$\boxed{\theta^* = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)}$$

Training neural networks

Let’s start easy

world's smallest neural network!
(a "perceptron")



$$y = wx$$

(a.k.a. line equation, linear regression)

Training a Neural Network

Given a set of samples and a Perceptron

$$\{x_i, y_i\}$$

$$y = f_{\text{PER}}(x; w)$$

Estimate the parameter of the Perceptron

$$w$$

Given training data:

x	y
10	10.1
2	1.9
3.5	3.4
1	1.1

What do you think the weight parameter is?

$$y = wx$$

Given training data:

x	y
10	10.1
2	1.9
3.5	3.4
1	1.1

What do you think the weight parameter is?

$$y = wx$$

not so obvious as the network gets more complicated so we use …

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

Modify weight w such that $\hat{y}$ gets **'closer'** to y

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

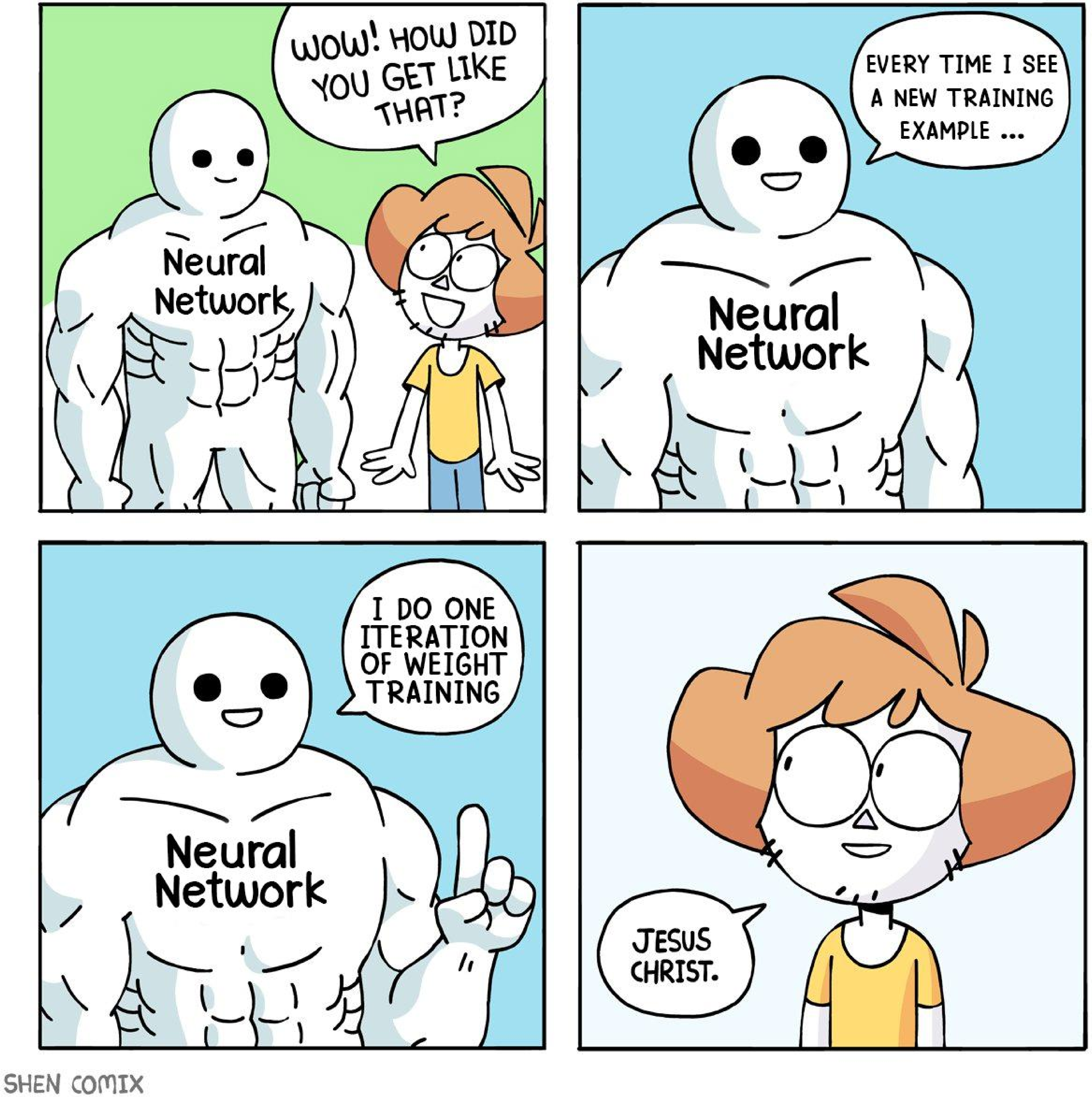
$$\hat{y} = wx$$

Modify weight w such that $\hat{y}$ gets **'closer'** to y

perceptron parameter

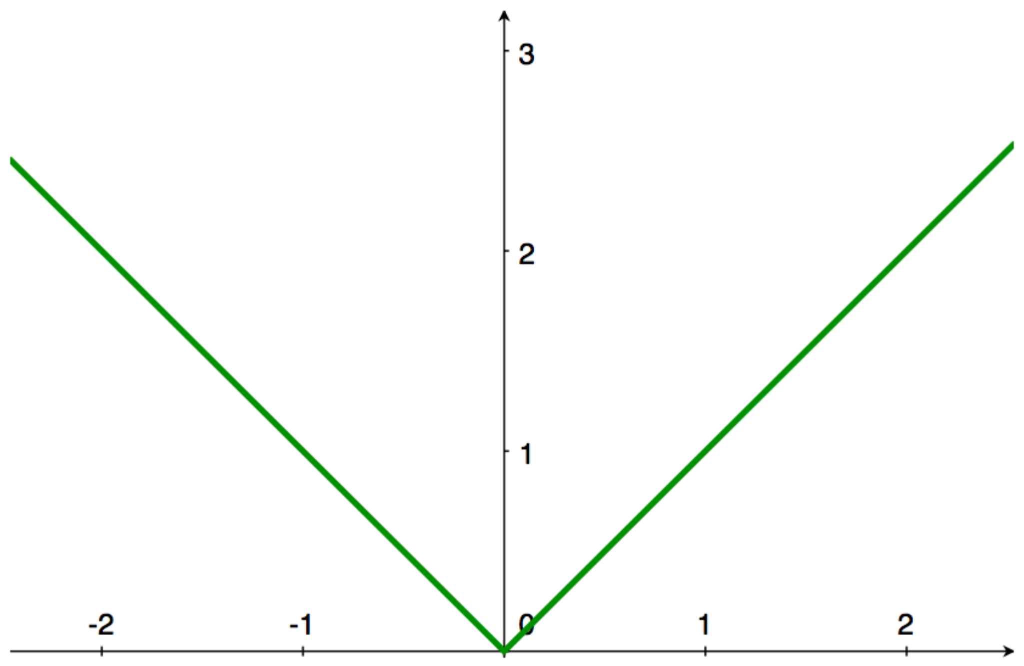
perceptron output

true label


WOW! HOW DID YOU GET LIKE THAT?
Neural Network
EVERY TIME I SEE A NEW TRAINING EXAMPLE ...
Neural Network
I DO ONE ITERATION OF WEIGHT TRAINING
Neural Network
JESUS CHRIST.
SHEN COMIX

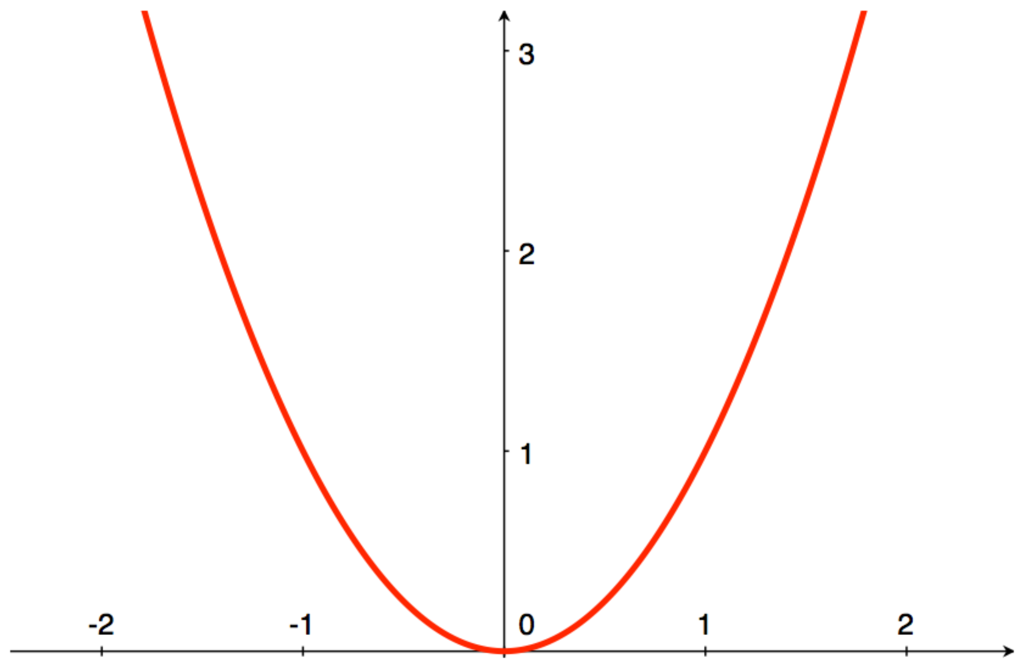
L1 Loss

$$\ell(\hat{y}, y) = |\hat{y} - y|$$



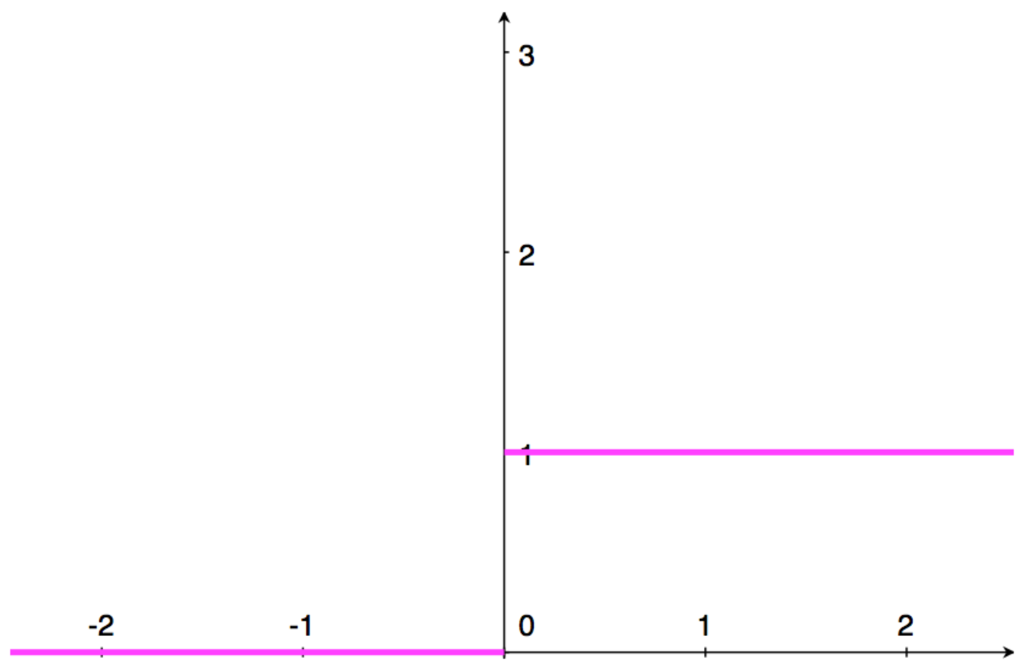
L2 Loss

$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$



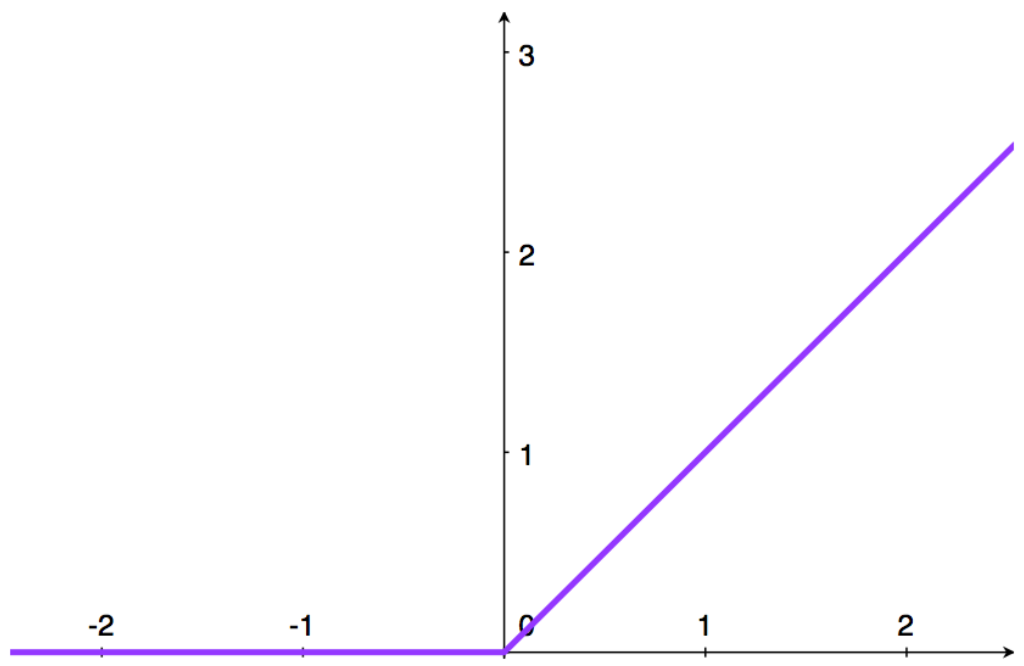
Zero-One Loss

$$\ell(\hat{y}, y) = \mathbf{1}[\hat{y} = y]$$

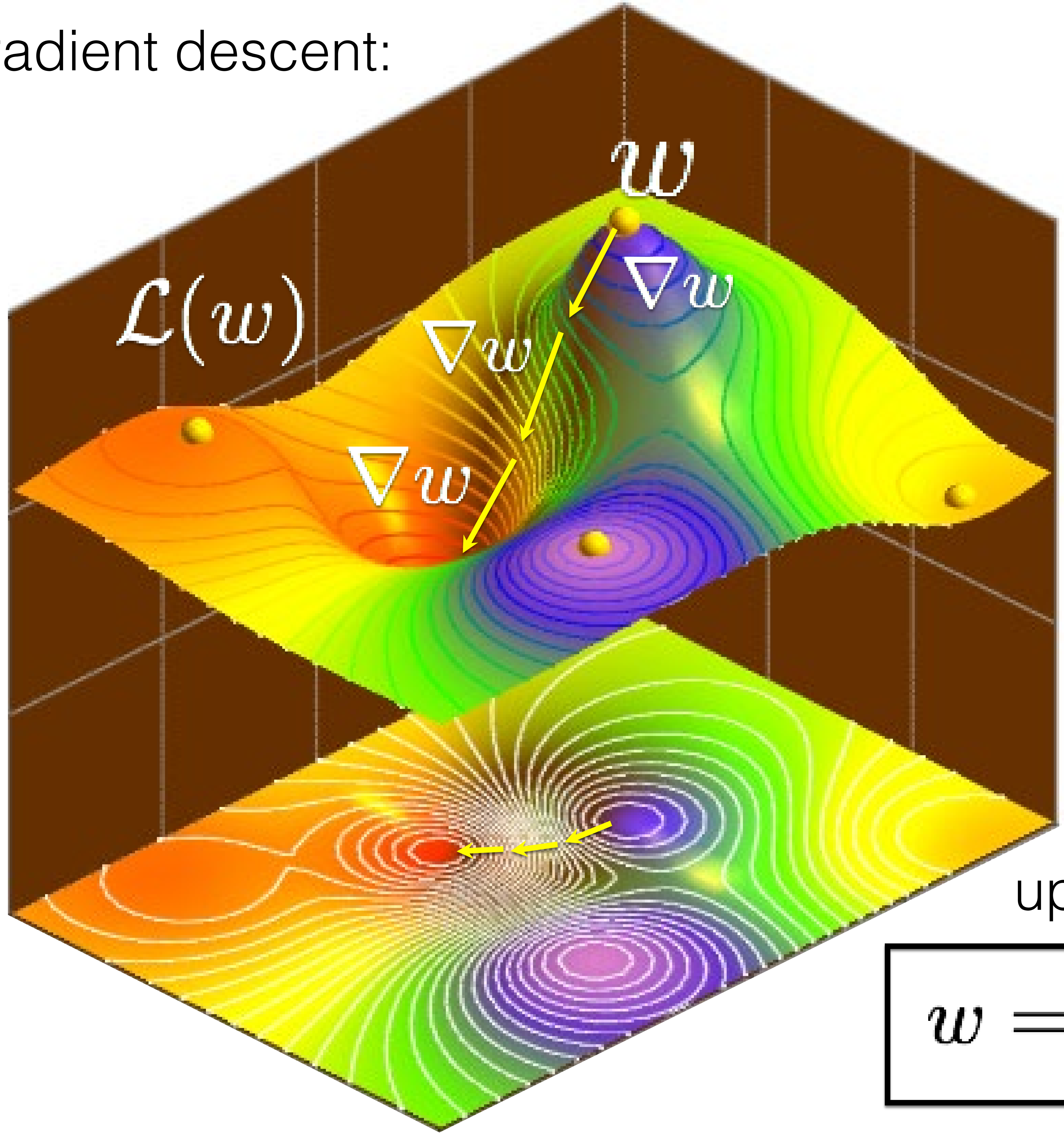


Hinge Loss

$$\ell(\hat{y}, y) = \max(0, 1 - y \cdot \hat{y})$$



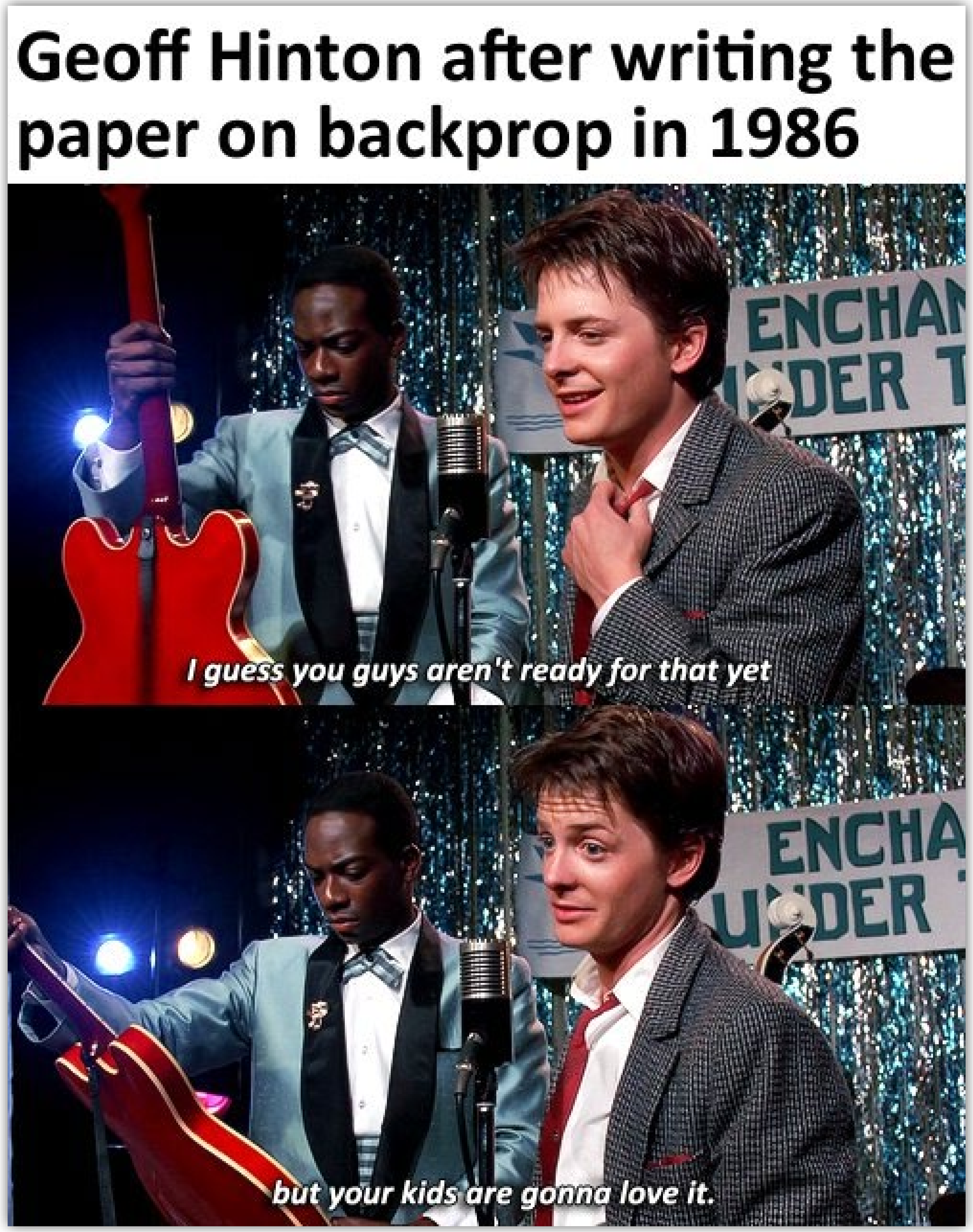
Gradient descent:



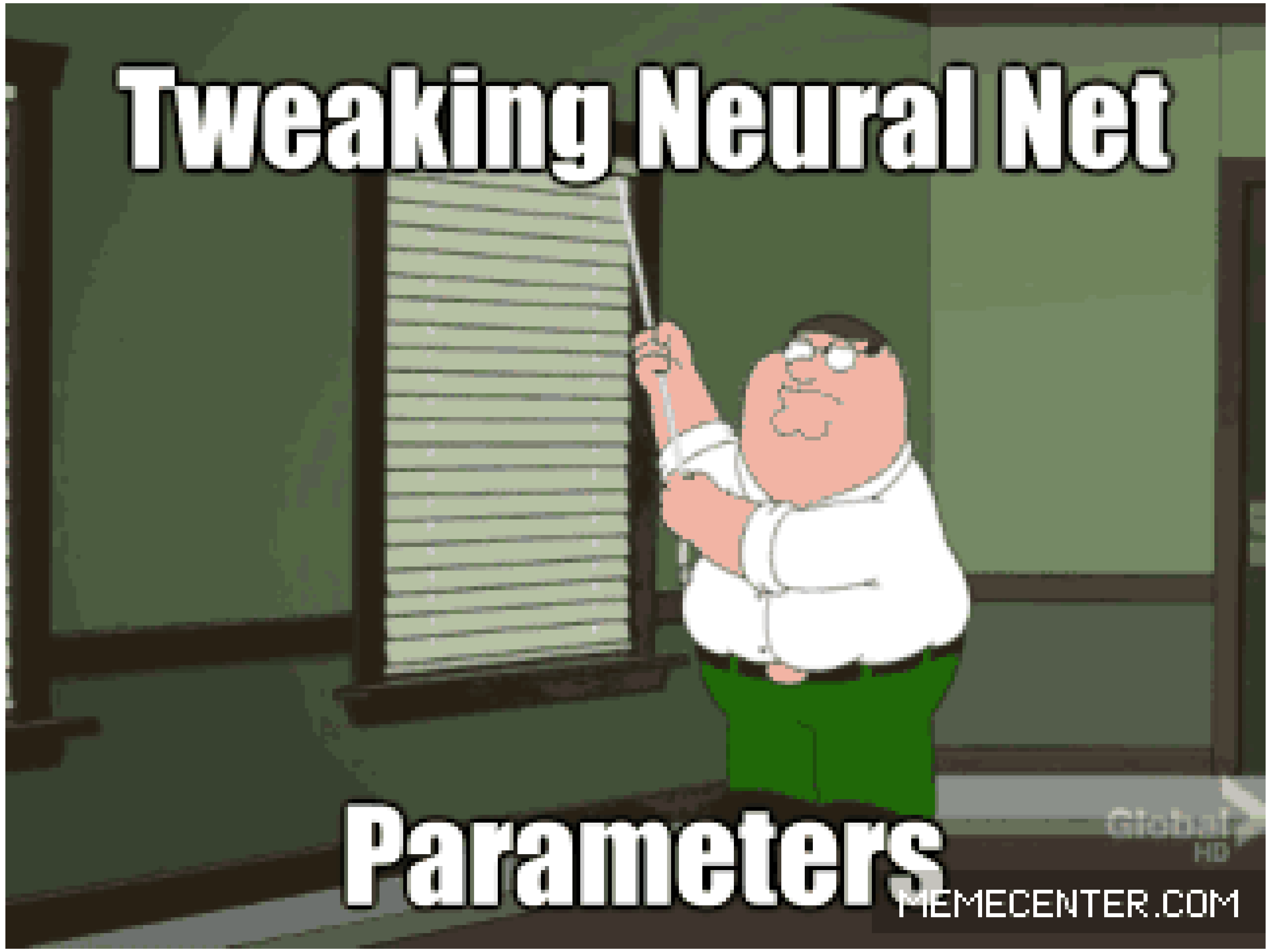
update rule:

$$\boxed{w = w - \nabla w}$$

Backpropagation



Backpropagation



$$\frac{d\mathcal{L}}{dw}$$ ...is the rate at which **this** will change...

$$\mathcal{L} = \frac{1}{2}(y - \hat{y})^2$$

the loss function

... per unit change of **this**

$$y = wx$$

the weight parameter

Let's compute the derivative...

Compute the derivative

$$\frac{d\mathcal{L}}{dw} = \frac{d}{dw}\left\{\frac{1}{2}(y-\hat{y})^2\right\}$$
$$= -(y-\hat{y})\frac{dwx}{dw}$$
$$= -(y-\hat{y})x = \nabla w$$

That means the weight update for **gradient descent** is:

$$w = w - \nabla w$$ move in direction of negative gradient
$$= w + (y-\hat{y})x$$

Gradient Descent (world's smallest perceptron)

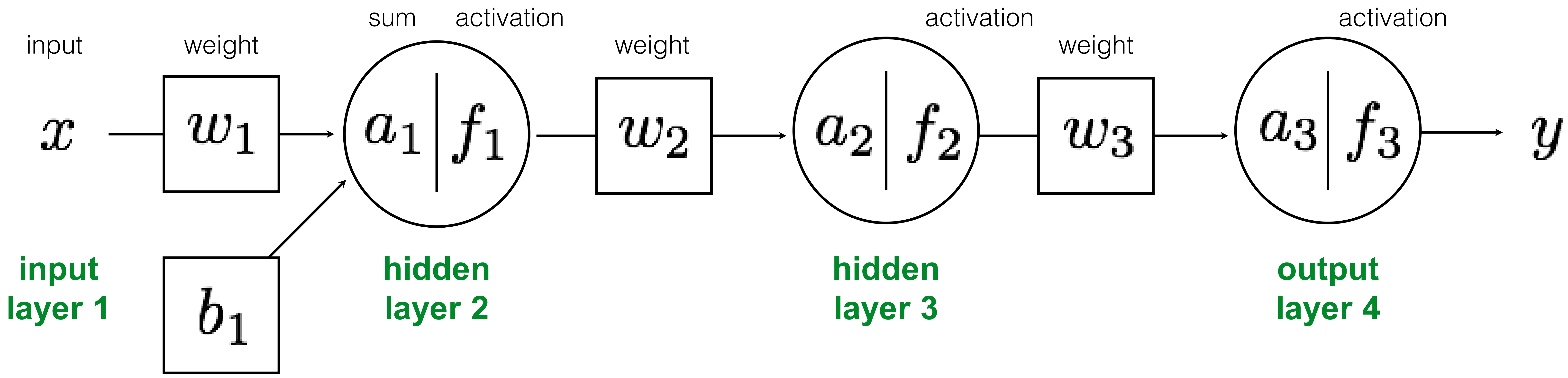
For each sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = wx_i$
   b. Compute Loss $\mathcal{L}_i = \frac{1}{2}(y_i - \hat{y})^2$
2. Update
   a. Back Propagation $\frac{d\mathcal{L}_i}{dw} = -(y_i - \hat{y})x_i = \nabla w$
   b. Gradient update $w = w - \nabla w$

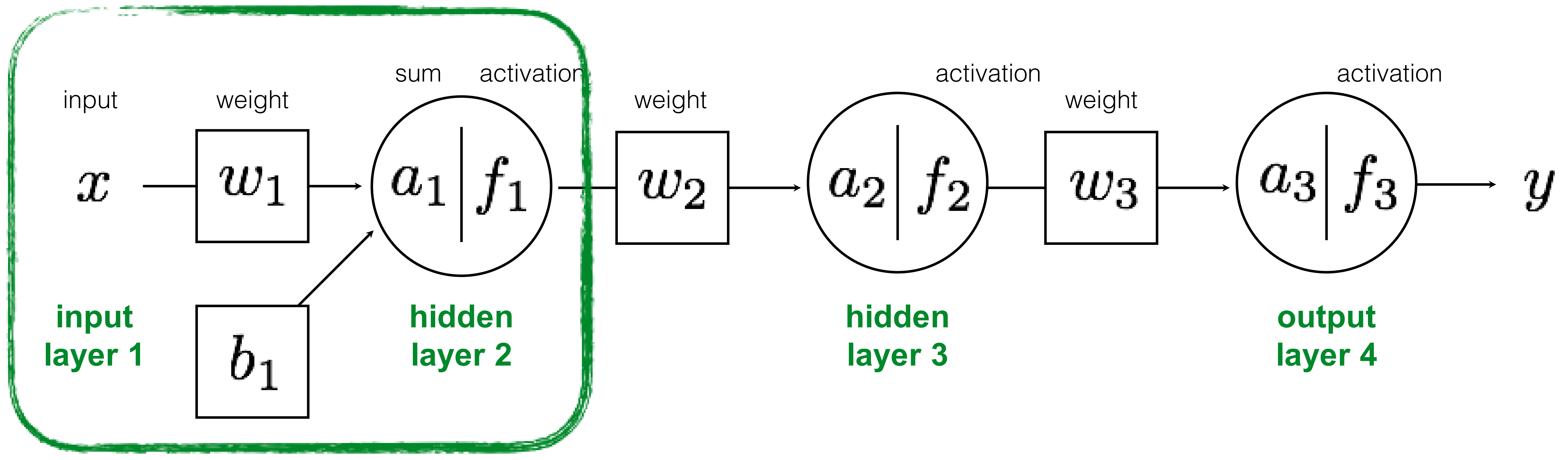
multi-layer perceptron

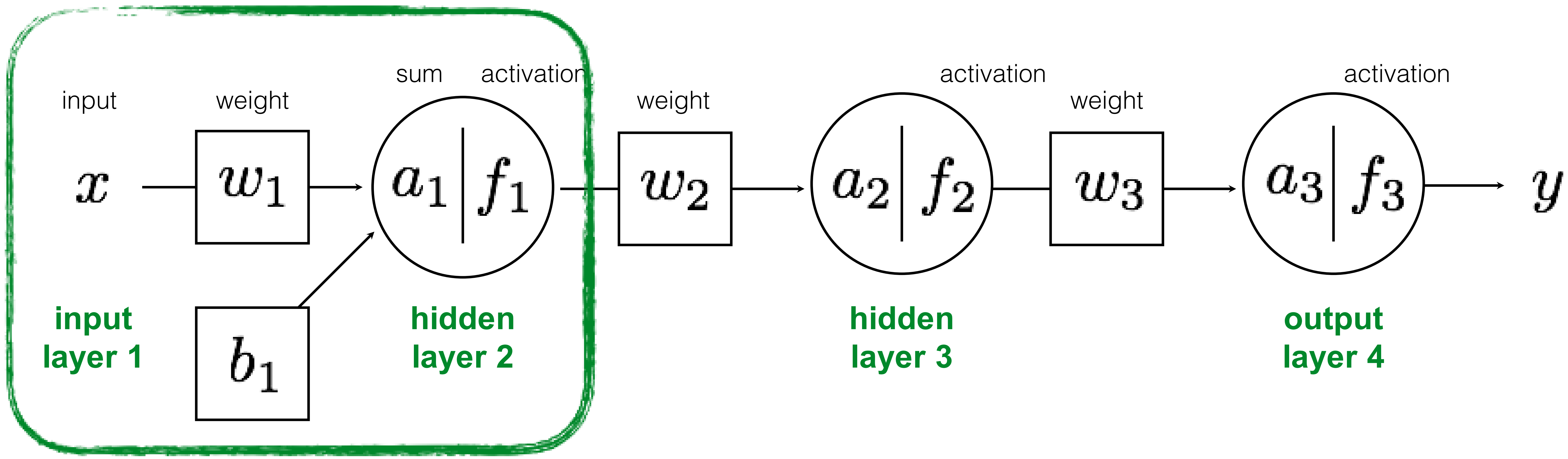
$$\begin{tikzcd} x \arrow[r, "w_1"] & h_1 \arrow[r, "w_2"] & h_2 \arrow[r, "w_3"] & o \arrow[r] & y \\ 1 \arrow[ur, "b"'] & & & & \end{tikzcd}$$

function of **FOUR** parameters and **FOUR** layers!

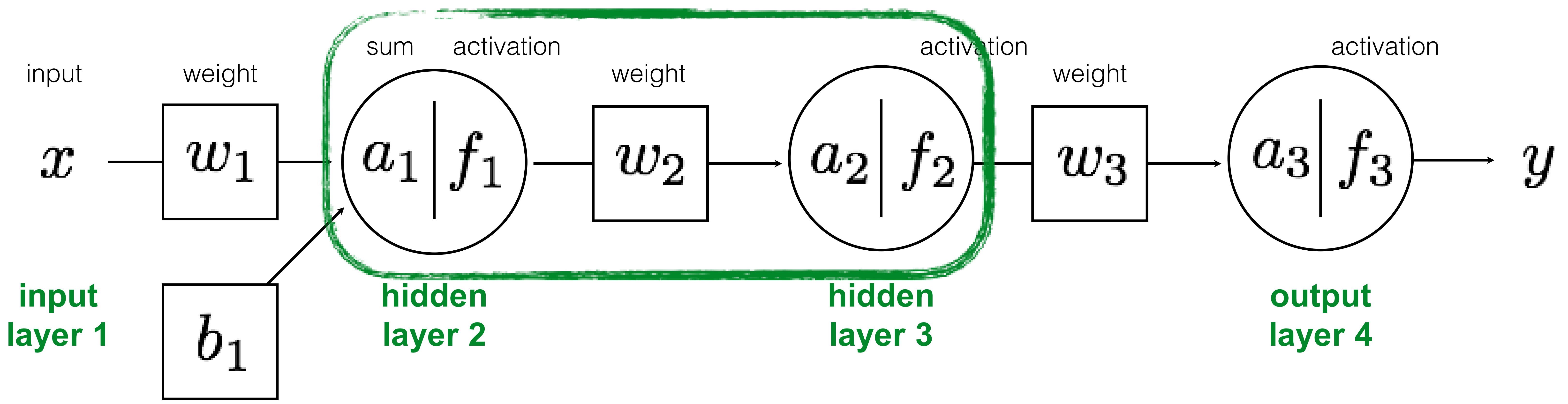


input
weight
sum
activation
weight
activation
weight
activation
x
w_1
a_1
f_1
w_2
a_2
f_2
w_3
a_3
f_3
y
b_1
input
layer 1
hidden
layer 2
hidden
layer 3
output
layer 4

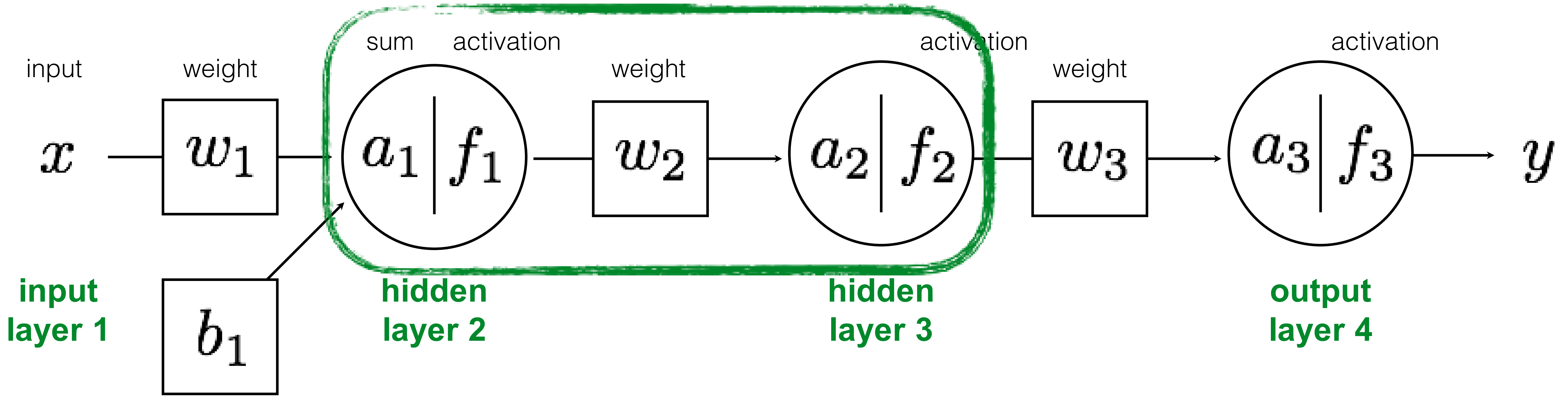

input
weight
sum
activation
weight
activation
weight
activation
x
w_1
a_1 f_1
w_2
a_2 f_2
w_3
a_3 f_3
y
b_1
input
layer 1
hidden
layer 2
hidden
layer 3
output
layer 4



$$a_1 = w_1 \cdot x + b_1$$

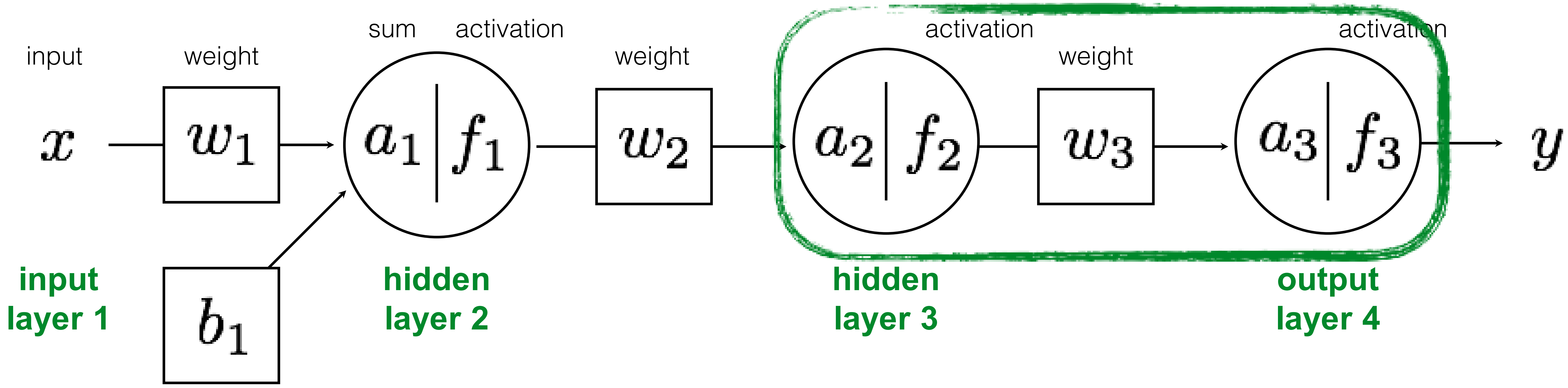


$$a_1 = w_1 \cdot x + b_1$$



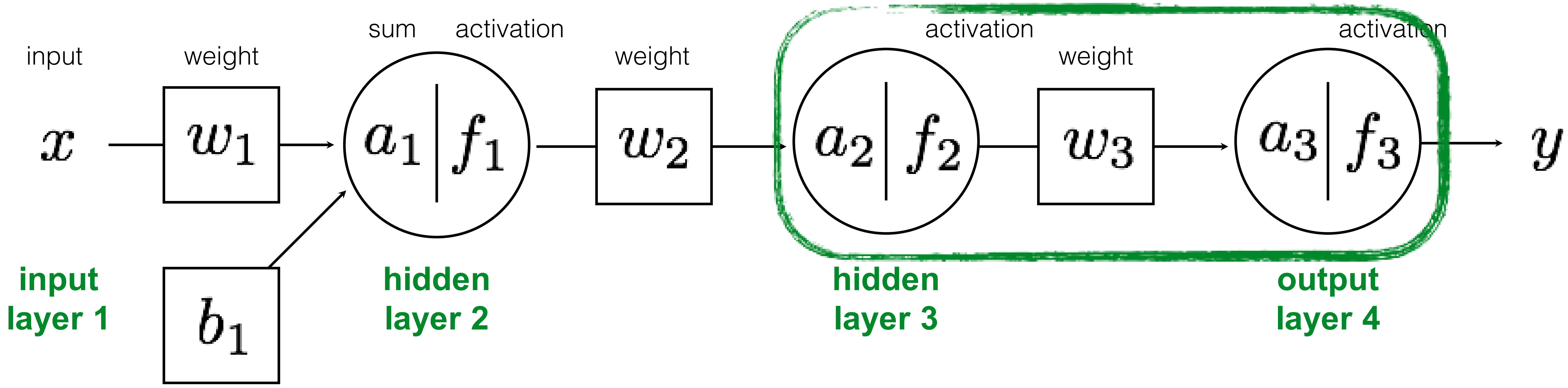
$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$



$$a_1 = w_1 \cdot x + b_1$$

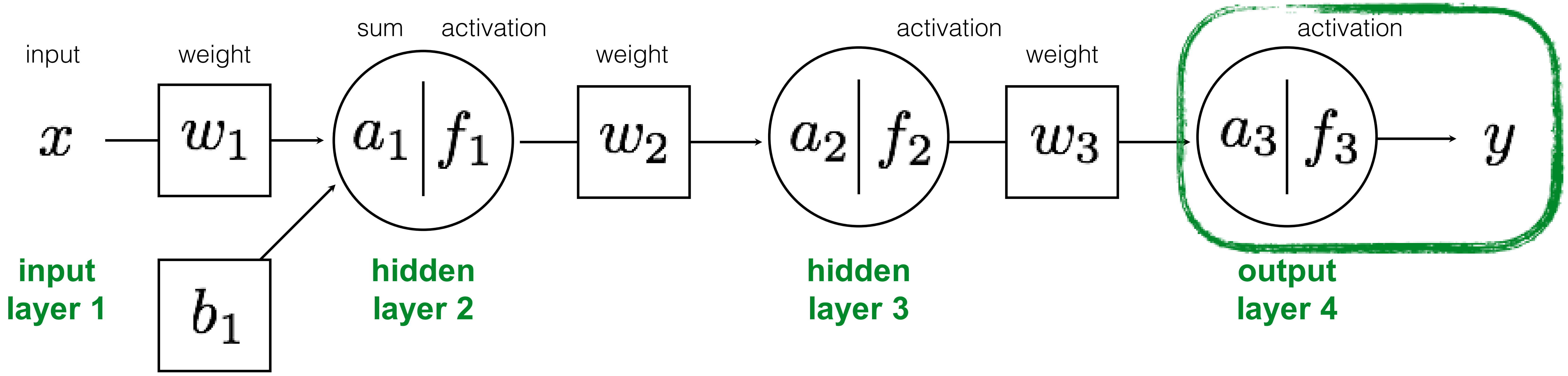
$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

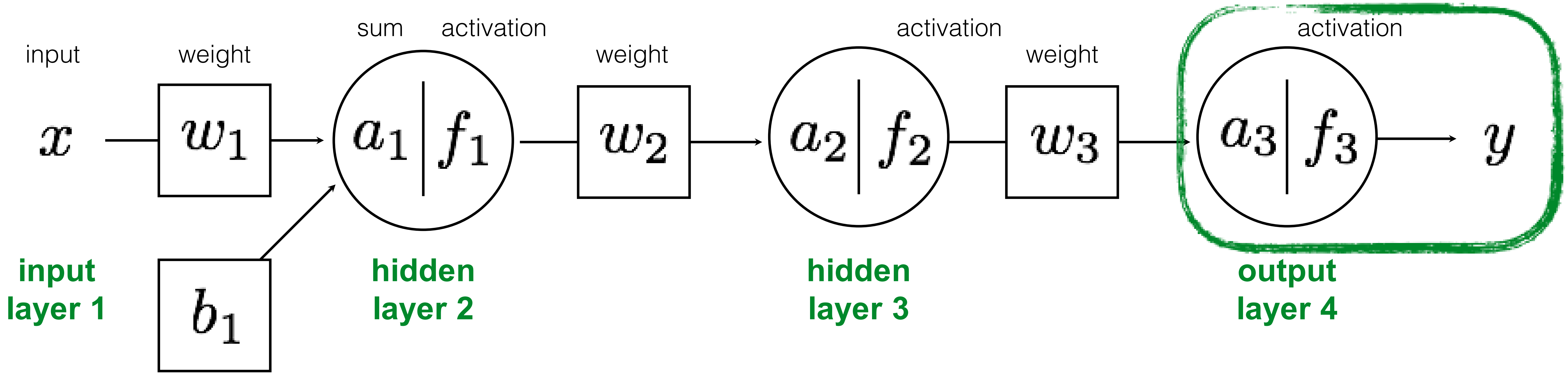
$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

Entire network can be written out as one long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

We need to train the network:

What is known? What is unknown?

Entire network can be written out as a long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

known

We need to train the network:

What is known? What is unknown?

Entire network can be written out as a long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

activation function
sometimes has
parameters

unknown

We need to train the network:

What is known? What is unknown?

Learning an MLP

Given a set of samples and a MLP

$$\{x_i, y_i\}$$

$$y = f_{\text{MLP}}(x; \theta)$$

Estimate the parameters of the MLP

$$\theta = \{f, w, b\}$$

Gradient Descent

For each **random** sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
   b. Compute Loss
2. Update
   a. Back Propagation $\frac{\partial \mathcal{L}}{\partial \theta}$ vector of parameter partial derivatives
   b. Gradient update $\theta \leftarrow \theta - \eta \nabla \theta$

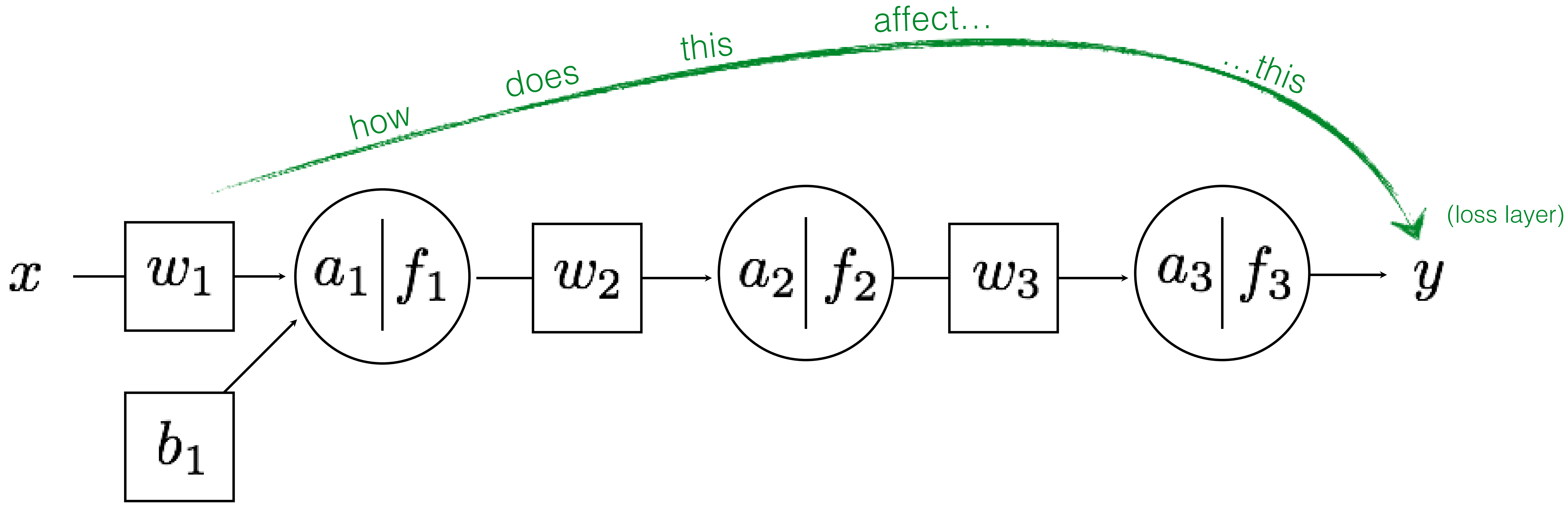
vector of parameter update equations

So we need to compute the partial derivatives

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \left[\frac{\partial \mathcal{L}}{\partial w_3} \frac{\partial \mathcal{L}}{\partial w_2} \frac{\partial \mathcal{L}}{\partial w_1} \frac{\partial \mathcal{L}}{\partial b} \right]$$

Remember,

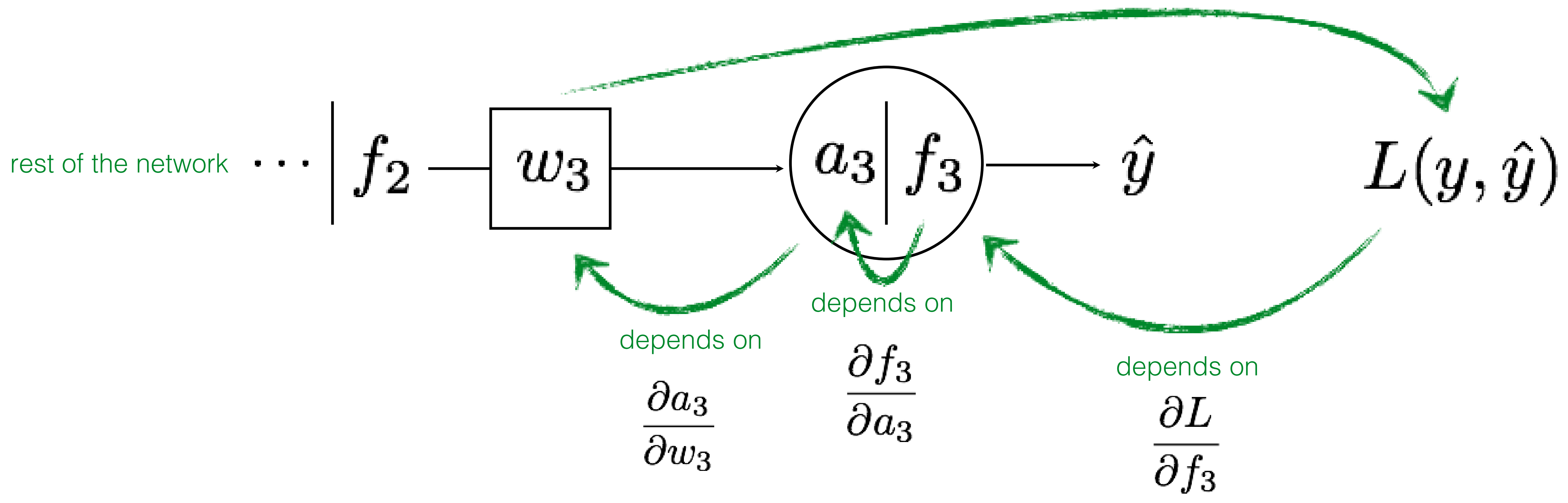
Partial derivative $\frac{\partial L}{\partial w_1}$ describes…

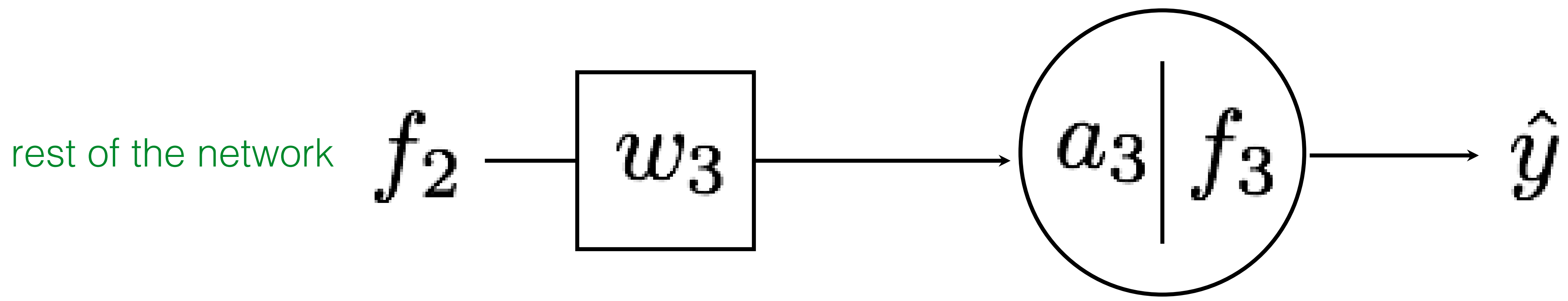


According to the chain rule…

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

Intuitively, the effect of weight on loss function : $\frac{\partial L}{\partial w_3}$





$L(y, \hat{y})$

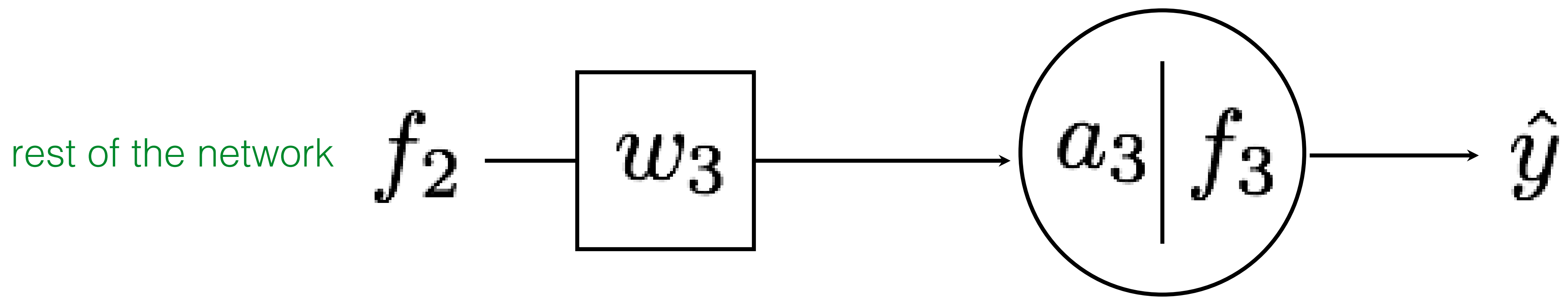
$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3}$$

Chain Rule!

rest of the network $f_2 \longrightarrow \boxed{w_3} \longrightarrow a_3 \,|\, f_3 \longrightarrow \hat{y}$ $\qquad L(y, \hat{y})$

$$
\begin{aligned}
\frac{\partial L}{\partial w_3} &= \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3} \\
&= -\eta(y - \hat{y}) \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3}
\end{aligned}
$$

Just the partial
derivative of L2 loss



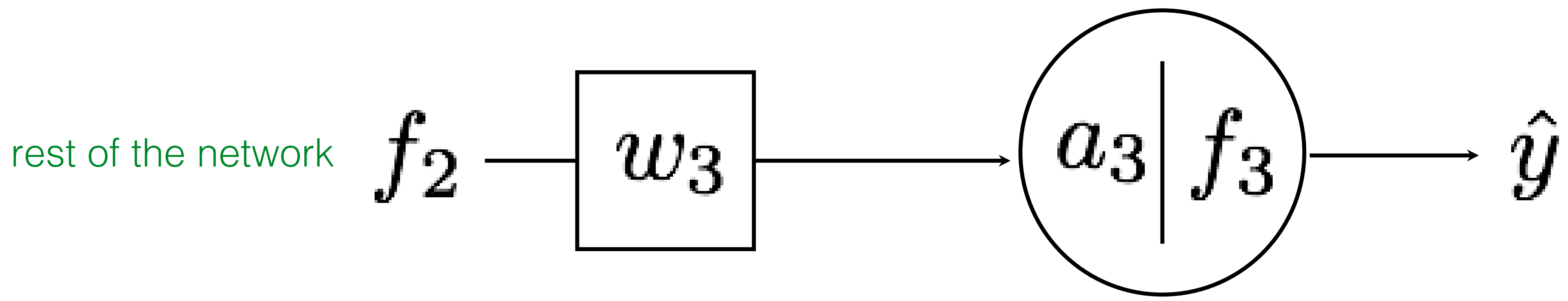
$$L(y, \hat{y})$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$= -\eta(y - \hat{y})\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

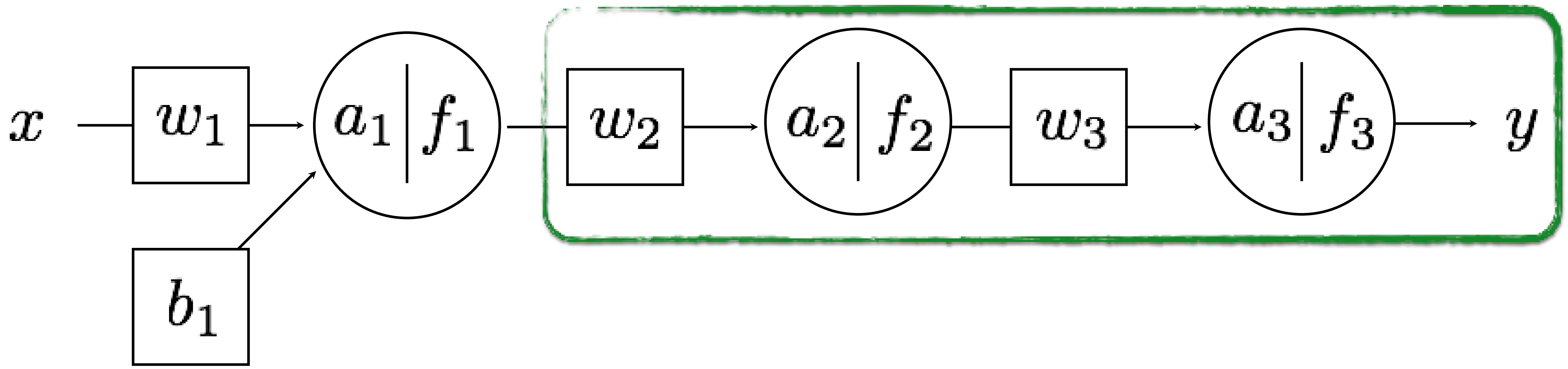
Let’s use a Sigmoid function

$$\frac{ds(x)}{dx} = s(x)(1 - s(x))$$

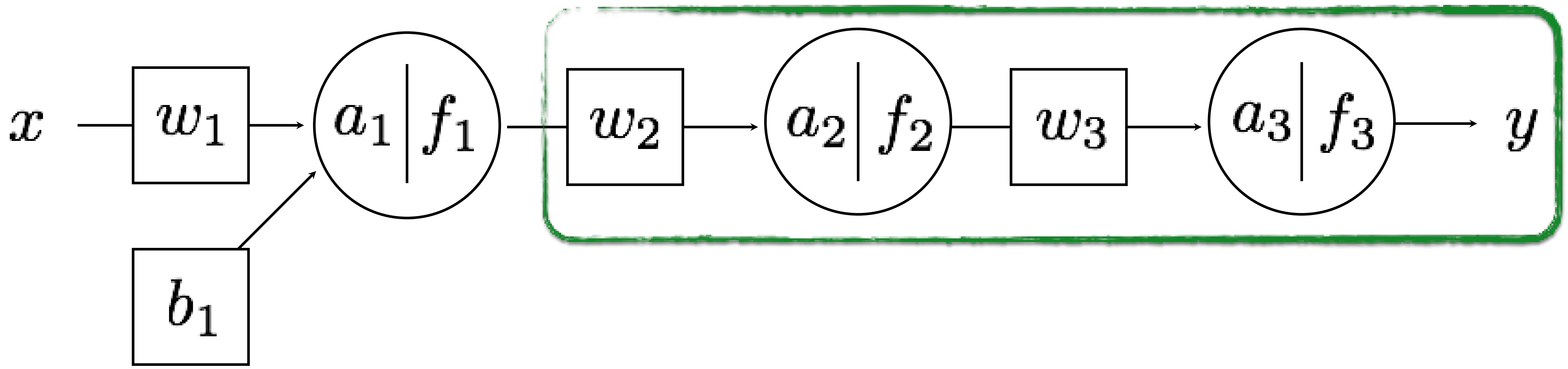


$$L(y, \hat{y})$$

$$
\begin{aligned}
\frac{\partial L}{\partial w_3} &= \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3} \\
&= -\eta(y - \hat{y}) \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3} \\
&= -\eta(y - \hat{y}) f_3(1 - f_3) \frac{\partial a_3}{\partial w_3} \\
&= -\eta(y - \hat{y}) f_3(1 - f_3) f_2
\end{aligned}
$$



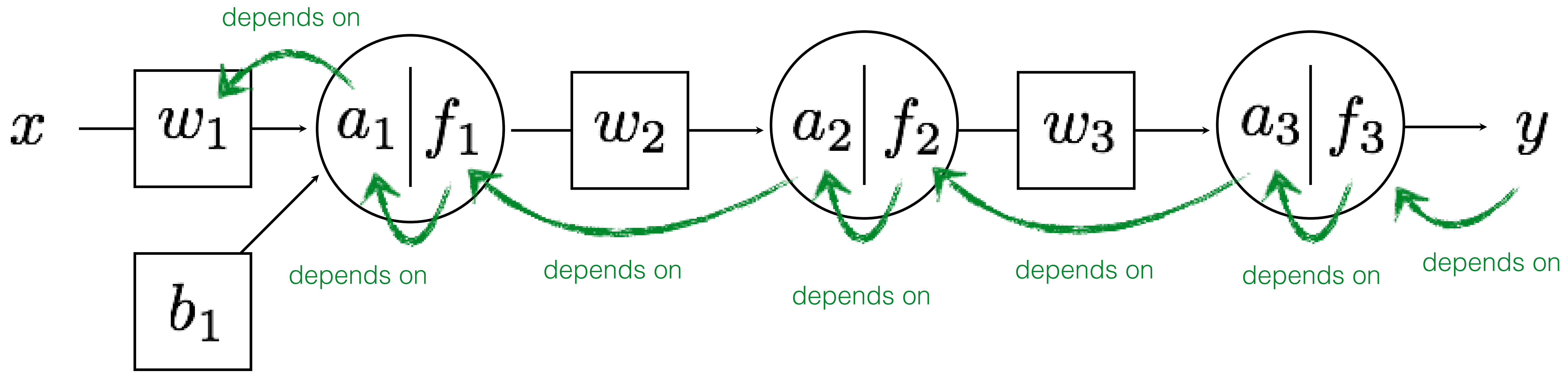
$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$



$$\frac{\partial L}{\partial w_2} = \boxed{\frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3}} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial w_2}$$

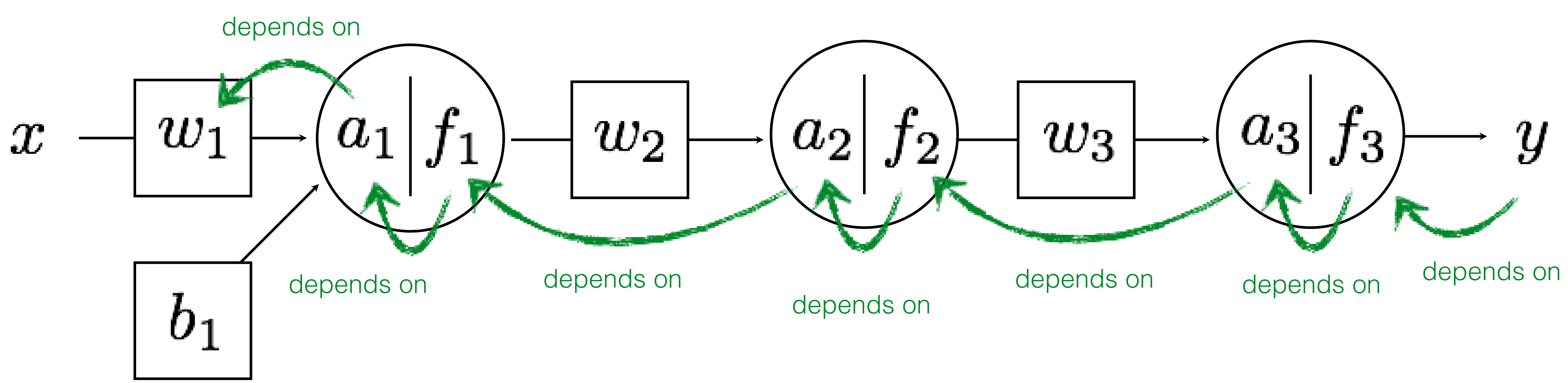
already computed.
re-use (<u>propagate</u>)!

The chain rule says…



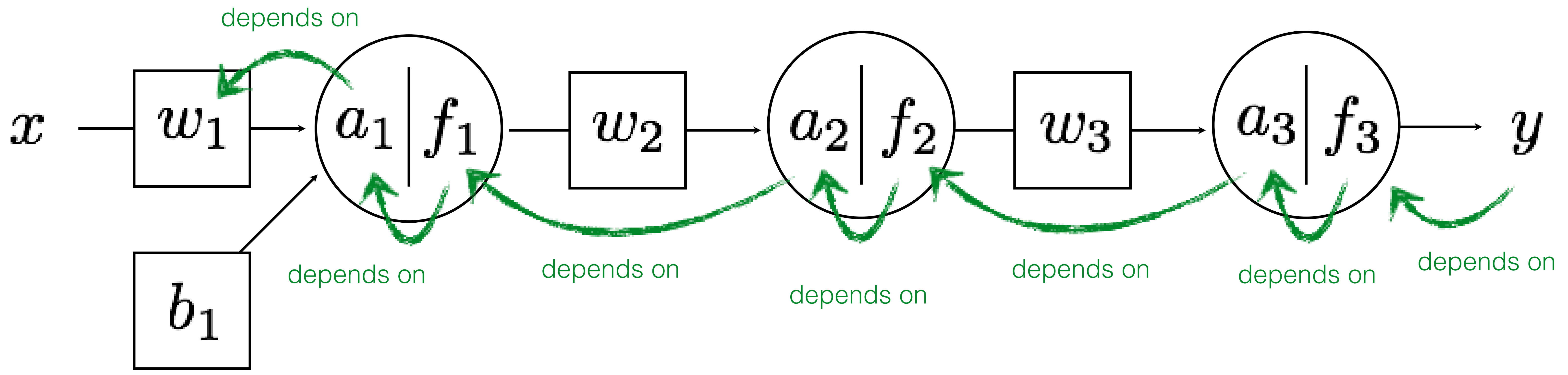
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

The chain rule says…



$$\frac{\partial L}{\partial w_1} = \boxed{\frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

already computed.
re-use (propagate)!

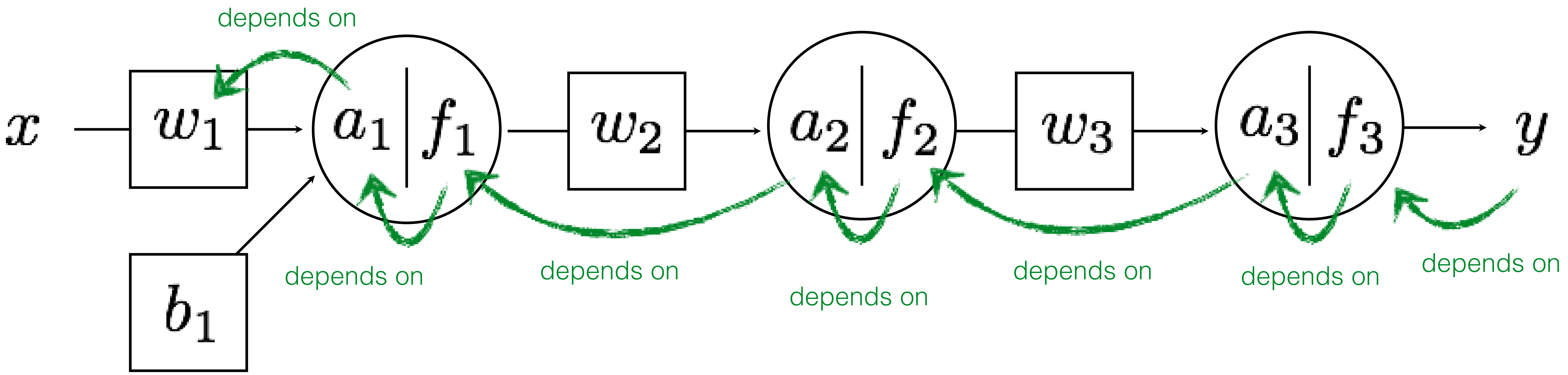


$$\frac{\partial \mathcal{L}}{\partial w_3} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}}\frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial b}$$

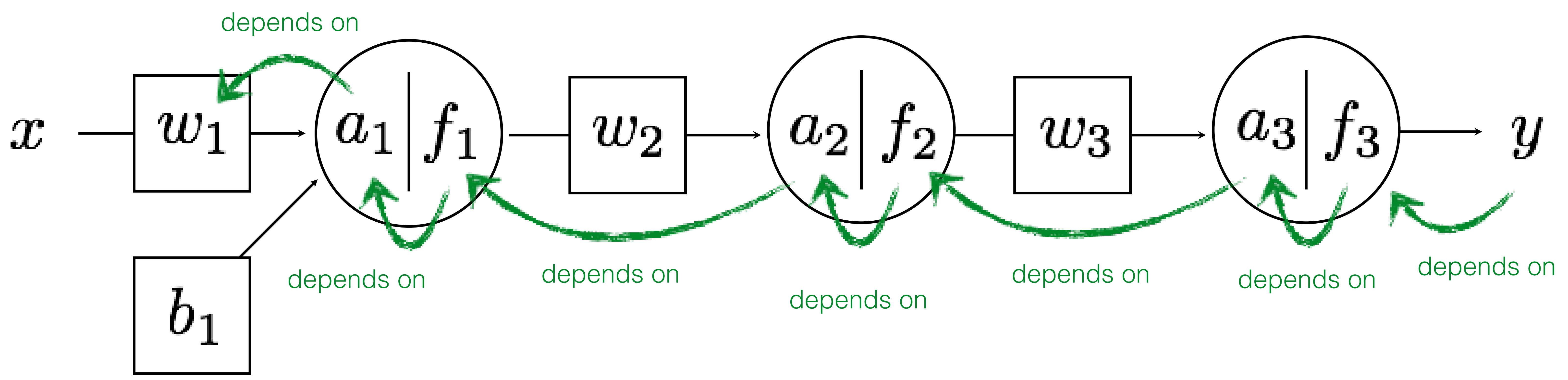


$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}}\frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial b}$$



$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial f_1} \frac{\partial f_1}{\partial a_1}} \frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial f_1} \frac{\partial f_1}{\partial a_1}} \frac{\partial a_1}{\partial b}$$

Gradient Descent

For each example sample $\{x_i, y_i\}$

1. Predict

   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$

   b. Compute Loss $\mathcal{L}_i$

2. Update

   a. Back Propagation

$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$
$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$
$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$
$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial b}$$

   b. Gradient update

$$w_3 = w_3 - \eta \nabla w_3$$
$$w_2 = w_2 - \eta \nabla w_2$$
$$w_1 = w_1 - \eta \nabla w_1$$
$$b = b - \eta \nabla b$$

Gradient Descent

For each example sample $\{x_i, y_i\}$

1. Predict

   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$

   b. Compute Loss $\mathcal{L}_i$

2. Update

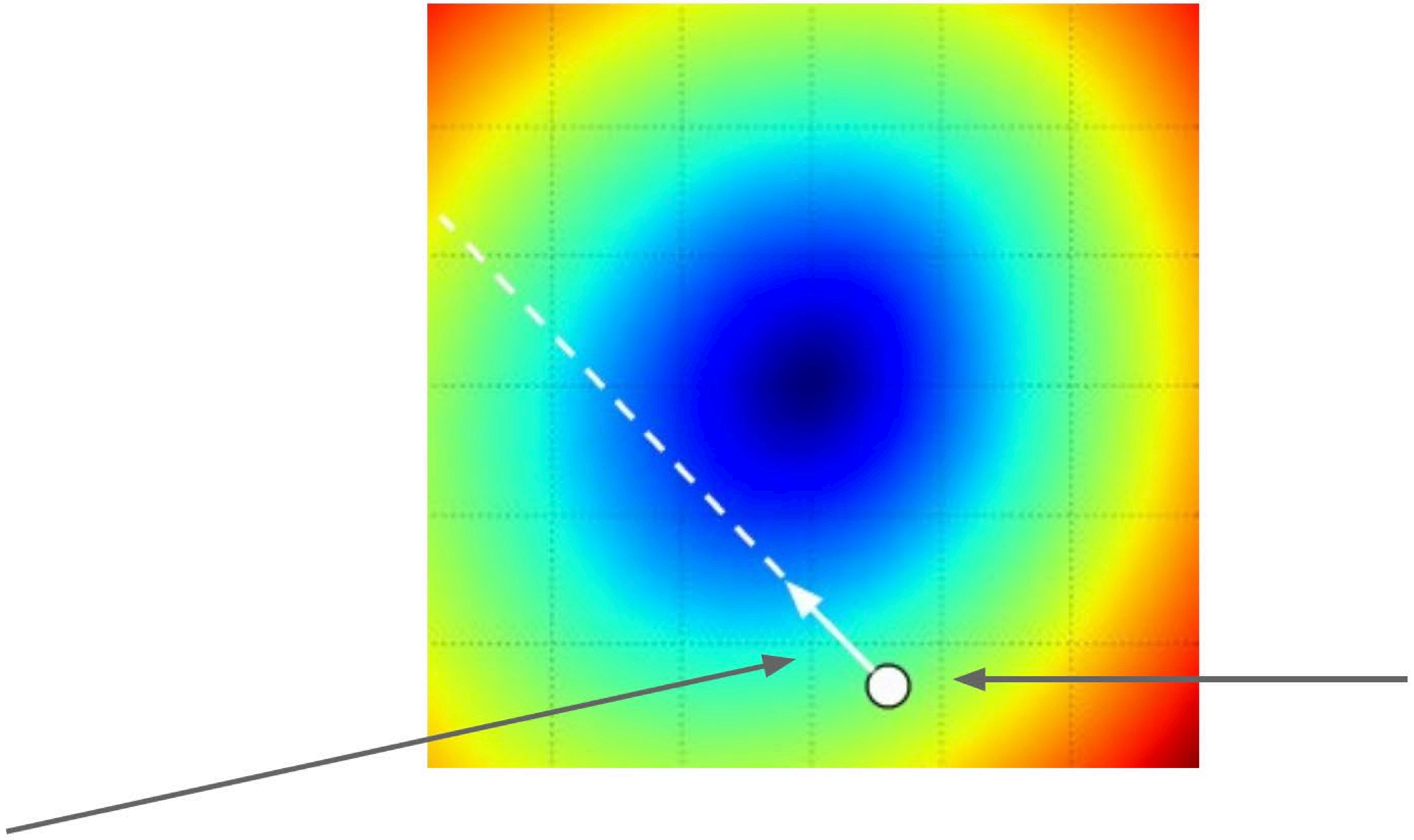
   a. Back Propagation $\dfrac{\partial \mathcal{L}}{\partial \theta}$

      vector of parameter partial derivatives

   b. Gradient update $\theta \leftarrow \theta - \eta \dfrac{\partial \mathcal{L}}{\partial \theta}$

      vector of parameter update equations

Learning rates



$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$$

Step size: learning rate
Too big: will miss the minimum
Too small: slow convergence

Gradient Descent

$$\nabla C = \left[\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right]^T \qquad \Delta v = \begin{bmatrix}\Delta v_1 \\ \Delta v_2\end{bmatrix}$$

$$\Delta C \approx \nabla C \cdot \Delta v$$

Let $\Delta v = -\eta \nabla C$

$$\Delta C = -\eta \|\nabla C\|^2$$